PAPER • OPEN ACCESS

A quantum active learning algorithm for sampling against adversarial attacks

To cite this article: P A M Casares and M A Martin-Delgado 2020 *New J. Phys.* **22** 073026

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

OPEN ACCESS

RECEIVED
23 December 2019

REVISED
15 May 2020

ACCEPTED FOR PUBLICATION
28 May 2020

PUBLISHED
20 July 2020

PAPER

A quantum active learning algorithm for sampling against adversarial attacks

P A M Casares[1] and M A Martin-Delgado

Departamento de Física Teórica, Universidad Complutense de Madrid, Spain
[1] Author to whom any correspondence should be addressed.

E-mail: pabloamo@ucm.es and mardel@ucm.es

Keywords: quantum algorithm, active learning, quantum machine learning, adversarial example, support vector machine, qRAM, quantum advantage

Abstract

Adversarial attacks represent a serious menace for learning algorithms and may compromise the security of future autonomous systems. A theorem by Khoury and Hadfield-Menell (KH), provides sufficient conditions to guarantee the robustness of active learning algorithms, but comes with a caveat: it is crucial to know the smallest distance among the classes of the corresponding classification problem. We propose a theoretical framework that allows us to think of active learning as sampling the most promising new points to be classified, so that the minimum distance between classes can be found and the theorem KH used. Additionally, we introduce a quantum active learning algorithm that makes use of such framework and whose complexity is polylogarithmic in the dimension of the space, m, and the size of the initial training data n, provided the use of qRAMs; and polynomial in the precision, achieving an exponential speedup over the equivalent classical algorithm in n and m.

1. Introduction

Supervised learning is one of the subareas of machine learning [1–3] that consists of techniques to learn to classify new data taking as example a training set. More specifically, the computer is given a training set X, consisting on n pairs of point and label, (x, y). With the information, the computer is supposed to extract or infer the conditional probability distributions $p(y|x)$ and use it to classify new points x. This paradigm is in contrast with unsupervised learning, that like in the case of clustering, attempts to find structure to a set of points without labels; and reinforcement learning [4], where an agent has to figure out the best policy or action for each situation it may face.

An important kind of supervised learning is what is usually called *active learning* [5]. To introduce this concept suppose that we have a supervised learning algorithm, with its corresponding training set. However, instead of directly trying to predict the label of new points, we give the classifier the option to pose us interesting questions in order to reduce the uncertainty in $p(y|x)$. In this setting, the algorithm will add to its training set new points in areas where it has a lot of uncertainty.

To explain the concept better, let us give an example. Suppose we have an image classifier used for a self-driving car, that has to to distinguish between cars, pedestrians…Internally, images are decomposed into pixels that can be characterised by their combination of red, blue and green. Thus, an image can be expressed as an array of 3 dimensional vectors, or as a large vector if we flatten the array. Any image is then a vector which can be identified with a point of a high dimensional space containing all images. The set of images of different classes forms a single manifold $\mathcal{M}$ embedded in that highly dimensional space. In particular, the dimension of the space is $3n_{\mathrm{p}}$, for n_{p} the number of pixels. The key idea of an active learning algorithm is one that is able to understand in what kind of images it has the most uncertainty, and request additional examples to be labeled and added to its training set.

Another important concept in the context of supervised learning is that of *adversarial attacks* or *adversarial examples*, the name given to a phenomenon where a trained and accurate (usually a neural

network) classifier, can be mislead into wrong classification, by producing a carefully chosen and slightly modified version of one point that is classified well. Adversarial examples were discovered quite recently [6, 7], and have received a lot of attention leading to models robust to particular attacks [8–11].

After the discovery of such adversarial examples, considerable effort has been put in explaining why they happen and how they can be avoided. One of the given theoretical reasons links their existence to a high codimension, the difference in dimension in the structure of the classes we are trying to separate with respect to the highly dimensional space in which they are embedded [12]. Intuitively this means that the dimension of the space of images of cars, for example, is much lower than the entire space of all possible images, of dimension $3n_{\mathrm{p}}$ as said earlier; since we need to impose a lot of constraints for an image to be indeed the image of a car, even if such constraints are not easily definable.

This suggest a strategy, proposed in theorem 5 of [12], that states that if we are able to cover our classes with a sufficiently fine sampling of the classes, our algorithm will be provably robust against these adversarial examples. However, how fine this sampling is depends crucially on the minimum distance r_p between classes, and since we do not fully know the classes, we also do not know this minimum distance with precision. Overestimating r_p will result in not being able to use theorem 5 of [12]; whereas underestimating it will mean oversampling the classes, with the associated cost.

In this article we present a quantum active learning algorithm that allows for fast sampling of the most informative points that could be added to the training set $X_{\mathcal{L}}$ to find out this minimum distance r_p. Here, the concept of 'informativeness' will refer to an expected gain of information, in the sense of improving the estimate of r_p. Our aim is to sample points to be added to the training set with the highest possible informativeness. It will be defined as the product of the probability that a given point is in a class, and the inverse of the margin of the quantum support vector machine, which measures the amount of information gained if the point were really in that class.

The quantum algorithm will allow us to perform this sampling very efficiently, in polylogarithmic time in the dimension of the space m and the number of already classified points, n, and polynomial cost in all other variables. To achieve such complexity we will need to use qRAMs [13] and techniques from [14, 15]. By comparison, linear algebra operations will require polynomial cost in n and m, when performed using classical computing.

The origin of this advantage is that in order to calculate such 'informativeness' we need to solve the quantum support vector machine, but we will not need to read out the solution. Rather, we will operate quantumly with the output to calculate the scalar value of 'informativeness', which we define formally in the next section. Notice that most quantum linear algebra algorithms are better suited for the cases where one does not have to read out the solution entirely, but rather calculate some expected value. This is partially the intuition that motivated our research.

We also hope that the framing of this article will highlight to the quantum machine learning community, often focused mostly on obtaining quantum advantages, the necessity of designing systems that are not only efficient and capable, but also robust and reliable.

The structure of the remaining sections of the article is the following. In the following section 1.1 we introduce related work on the topics of adversarial examples, active learning, and quantum machine learning; we also briefly mention the differences between adversarial examples and generative adversarial networks. In section 2 we introduce a bit more of background on theorem 5 of [12], and explain how to model the problem of finding the minimum distance between classes as an active learning problem. In particular we introduce the important concept of 'informativeness'. Finally, some background on support vector machines is reviewed in section 3.

Section 4 constitutes the main corpus of the text. To perform the active learning algorithm, in section 4.1 we explain how to obtain $P_c(\vec{x})$, the probability that an arbitrary point of the space is in a given class, which is one of the main components of the 'informativeness' of such point. The other main component is described in the next two subsections. In section 4.2 we review the main results that we will be using from our reference [14], and in section 4.3 we use the result of the previous section to calculate $|\vec{w}|_{n+1}$, the second main component of the 'informativeness' of $\vec{x}$. Section 4.4 explains the strategy to select a point that with high probability improves the current estimate of the SVM. Section 5 reviews the main results, and section 6 is dedicated to the calculation of the complexity of our algorithm. Finally, in section 7 we explain our conclusions. In the appendices we include definitions, theorems from other articles that we use, and some technical results.

1.1. Related work

Adversarial examples are a danger for any classifier that needs to be robust to perturbation. For instance, adversarial examples can be dangerous when an autonomous car has to recognise traffic signals. Since adversarial examples were discovered [6], there has been lots of work to explain why they happen [7] and

also to obtain provably robust models [8]. In particular some of the most promising ideas to avoid them are related to adversarial training: training against those adversarial examples before the actual adversary has time to pose them to the classifier. This is for instance the model explained in [8], where they use projected gradient descent to minimize the maximum expected loss

$$\min_{\theta}\left(\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\epsilon\in\mathcal{S}} L(\theta, x+\epsilon, y)\right]\right), \tag{1}$$

where $\mathcal{D}$ is the initial population of points x, with true label y, and perturbations ϵ can be taken from a small set $\mathcal{S}$. L is the loss function: the function that measures the difference between the predicted and actual classification, with adjustable parameters θ, and $\mathbb{E}$ indicates expected value. The maximisation represents the work of the adversary, whereas the minimization represents the work to make the classifier robust. This setup certainly works, as long as one can minimize (1), and efforts have been put forward to perform this adversarial training more efficiently [16]. However, as pointed out in [12], in order to work perfectly it would require an exponential number of adversarial examples in the dimension of the problem added to the training set. Thus, additional strategies are worth exploring.

It is also worth noticing that adversarial attacks are gaining attention in the quantum community. Recently, it has been indicated that this phenomenon is also present in the case of quantum classifiers [17], where the dimension plays a very important role: the higher the dimension the easier to carry out those adversarial attacks. Some experimental work on this line is done in [18].

Our work is additionally strongly related to several forms of *active learning* algorithms. Active learning algorithms are those where the classifier can ask for new points to be classified and added to its training data base. This field can be divided in two main branches [19]: *query synthesis*, where new examples are created, and *sampling*. The later is subdivided in *stream-based sampling* and *pool-based*. In stream-based sampling one selects one item at a time and decides whether it is worth the cost of classifying it [20]. In pool-based sampling examples are sampled from a large pool of unlabelled data [19]. As we will see, our algorithm can be used both as a pool-based sampling or as query synthesis, depending on the points used. The most typical strategies to select the samples are *uncertainty sampling*, that selects points with maximum uncertainty about which class they belong to [21], or *Query-by-committee*, where the space of classifiers that agree with the data is halved sequentially [22]. Finally there is also the strategy of using *expected error reduction* [23] that selects those points that on average, weighted according to probability, make the loss function as low as possible. This last procedure is similar to what we are using, except that instead of minimizing the loss function, we select points that on expectation would achieve the margin of the SVM to be as low as possible.

Finally, let us make a brief introduction to quantum algorithms used for quantum machine learning, since the quantum SVM that we have used through the text is just one of the earlier works on this field. The possibility of applying quantum techniques in the machine learning and artificial intelligence domain have attracted in the last decade much interest, and a review of this efforts is [24]. For example, following work on quantum SVM, various kernel methods have been developed to improve support vector machines, such as [25] or [26]. Most of this efforts have been focused on approaches that can be implemented NISQ devices. For instance, the later reference, [26], was specially focused on constructing a universal quantum classifier with minimal amount of quantum resources, using a technique called data re-uploading.

Other machine learning techniques that have been quantized include Bayesian learning [27], where a polynomial speedup is expected thanks to the use of linear algebra techniques such as variations over HHL.

Finally, although considerable effort has been put on quantizing machine learning algorithms such as support vector machines and kernel methods, given the success of the deep learning (aka neural networks) techniques in tackling many problems that include image recognition and text translation, significant effort has be put in quantizing them. One example of this is [28], or the recent publication of tensorflow quantum [29], the tensorflow library to work with quantum neural networks. One particular example of a deep learning technique that has received much attention both in the classical and quantum machine learning communities are the generative adversarial networks [30]. This networks are composed of two parts, a generator and a discriminator that compete, and a distribution $\mathcal{D}$. The generator tries to generate a distribution resembling as much as possible $\mathcal{D}$, and the discriminator tries to differentiate between the actual $\mathcal{D}$, and the one generated by the generator. The game ends when the generator has fully learned $\mathcal{D}$, and precisely this is the reason why adversarial generative networks have been devised as a method to prepare quantum states efficiently.

However, we would like to emphasise that although somewhat related to adversarial examples via adversarial training, the setup in the later case is different. For example, in the setting of adversarial examples our classifier does not need to know how to generate any state of the distribution, but only how to classify an input in one of the classes. In such respect, there is no generative device in adversarial examples,

except perhaps the adversary who is trying to fool the classifier, whereas in the generative adversarial training setup there is a single distribution, and therefore a single class. Perhaps more importantly, adversarial examples seem to be a general and somewhat unfortunate feature of many different kinds of classifiers, whereas generative adversarial networks are a particular, although useful, technique.

2. Modelling the active learning problem

As mentioned in the introduction, the problem we want to solve in this article is the following:

Problem 1 Informative points sampling problem. *Suppose we are given a set of points* $S = (\vec{x}_i, \vec{c}_i)$, $i \in \{1, .., n\}$, *where* $\vec{x}_i \in \mathbb{R}^m$ *are the n data points already classified, and* $\vec{c}_i$ *are label vectors that indicate probability of membership to the possible classes where the point might be classified. Suppose also that these memberships sum up to, at most, 1 for each point, and for simplicity assume that there are just two classes. The problem is: what are the most informative points to be added to S in order to learn the support vector machine, and in particular the minimum distance between classes* r_p, *with better precision?*

This problem makes reference to two key concepts: support vector machine and 'informativeness', that we introduce now. Although the technical definition of support vector machine (SVM) is stated in the section 3, it is basically the hyperplane of separation between two classes that maximizes the margin.

In our particular problem, we are interested in finding the minimum distance between two classes, in order to provide the required covering of the two classes that avoid having adversarial attacks. However, it might be the case that both classes have some points in common, e.g. $(\vec{c}_i)_j, (\vec{c}_i)_k > 0$, where i indicates the point and j and k two classes. In order to avoid this, we establish that a point $\vec{x}_i$ is in class j if $(\vec{c}_i)_j > 0.8$, and as $\sum_j (\vec{c}_i)_j = 1$ for any point $\vec{x}_i$, this will avoid any overlap between classes. This value is arbitrary but should be strictly greater than 0.5. This way we clearly separate the two sets, and the minimum distance between classes is greater than 0. With this condition one wants to make classes clearly separated, since otherwise it does not make sense the concept of adversarial example, which will be central to our discussion. If there are more than two classes, then one can generalise the previous setting making comparison on all possible pairs of classes, which scales quadratically with the number of classes.

Now let us look at figure 1. We can see that the initially calculated SVM has a greater margin than that of the SVM we would have if we were able to perfectly know both classes. If we used such margin in theorem 1, we would not find a cover that avoids adversarial examples. Thus, we are interested in minimizing the maximum margin, which is given by the true SVM if we knew the actual classes and not just some samples.

Our problem is however in contrast with usual pool-based sampling (see section 1.1 for more information). In their problem the algorithm usually seeks to classify new points near the SVM. This is not the case for our problem, since we want points that have a membership to a class greater than 0.8. Points that have no clear membership to a single class are not so interesting and we do not include them in our data set.

Now we turn to the concept of 'informativeness'. How then do we measure how interesting could be to classify an arbitrary point? A good heuristics for our problem is that we are trying to find points that with high probability would decrease the margin $1/|\vec{w}|$ a lot, to get a better estimate of r_p. This is because there are two competing conditions on r_p. On the one hand, in order to fulfill the condition of the problem, we cannot take r_p larger than it really is, as that would make us choose a cover of the classes that does not fulfill the conditions of the theorem 5 of [12]. On the other, the smaller r_p is, the more expensive it is to establish the cover of the classes.

The previous paragraph suggests measuring the 'informativeness' of a point by dividing the problem in two parts: we first find the probability that a given point $\vec{x}_{n+1}$ is in class c, and then multiply this probability by the inverse of the margin distance of the updated SVM, taking into account this would-be newly classified point, $|\vec{w}_{n+1}|$.

The 'informativeness' will thus measure an expected value of information gain if a given point was in a given class. As such, it will be the product of a probability of $\vec{x}$ being in class c, $P_c(\vec{x})$, times the information we gain, that we will measure as $|\vec{w}|$. The reason for this choice is that we want to add points to training set, that give a more precise account of the classes. That is, we wish to minimize the margin of size $1/|\vec{w}|$, and therefore we want to maximize $|\vec{w}|$.

Definition 1. '*Informativeness*' measures the expected value of information we get by adding a point $\vec{x}_i$ to the training set. We will therefore define it as the product

$$P_c(\vec{x}_i) \cdot |\vec{w}_{\vec{x}_i}|, \tag{2}$$

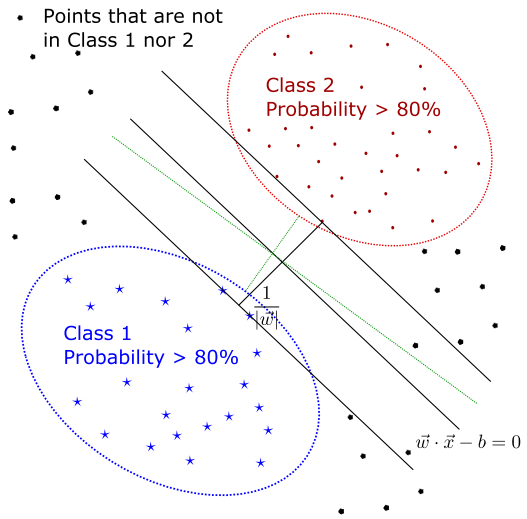

Figure 1. Example of an SVM, characterised by the equation $\vec{w}\cdot\vec{x}-b=0$. The points depicted are the initial training set, with colors according to their classification, and those in black are those not belonging to any class according to our criteria that for a point $\vec{x}_i$ to be in class j, $c_{ij} > 0.8$. Initially we have the SVM in black, but we want to obtain the green one, that is more accurate. The dotted lines are the ones that we cannot see initially and must find out. The equation of the SVM is the one depicted and the margin $1/|\vec{w}|$ is chosen to be equal to r_p. This is achieved when we make the two parallel hyperplanes that indicate the margin to fulfill equations $\vec{w}\cdot\vec{x}-b=\pm 1$. Since making the SVM more precise implies making the margin smaller, we want to find an SVM that maximizes $|\vec{w}|$.

where $P_c(\vec{x}_i)$ is the probability that $\vec{x}_i$ is in class c, and $1/|\vec{w}_{\vec{x}_i}|$ is the size of the margin of the resulting SVM if $\vec{x}_i$ really were in such class.

3. Background on support vector machines

Since we will rely on them somewhat heavily, in order to carry out those results it is useful to remember some results from [14], that explains how a quantum support vector machine algorithm works. Let us first introduce the definition of a support vector machine.

Definition 2 Support vector machine (SVM). Let $\mathcal{M}$ be an m-dimensional manifold with two submanifolds or classes and a set of points already classified $\{\vec{x}_i, y_i\}$, $\vec{x}_i$ being the point and y_i the label. Then a *support vector machine* is a hyperplane in the manifold separating both classes such that minimum distance between $\{\vec{x}_i\}$ and the hyperplane is as large as possible. If the SVM is linear it may be described by equation

$$\vec{w}\cdot\vec{x}-b=0, \tag{3}$$

and the size of the margin is $1/|\vec{w}|$, which would be equal to $r_p/2$ if the SVM was perfect. One may also define a non-linear SVM using a non-linear kernel for the dot product.

We highlight that the name support vector machine (SVM) will refer to both the algorithm and the separation hyperplane, the decision boundary.

The authors of [14] assume that each point $\vec{x}_i$ is labeled with a single class y_i, and there are only two classes $y_i=\pm 1$. Given the pairs of data and label $(\vec{x}_i, y_i)_{i\in\{1,\ldots,n\}}$, the authors state that calculating the SVM is equivalent, in the dual formulation, to maximizing over the multipliers $\vec{\alpha}$ of the Lagrangian

$$L(\vec{\alpha})=\sum_{i=1}^{n}\alpha_i y_i-\frac{1}{2}\sum_{i,k=1}^{n}\alpha_i K_{ik}\alpha_k, \tag{4}$$

with constraints $\sum_i \alpha_i=0$ and $\alpha_i y_i \geqslant 0\,\forall i$. After this, the result of the classifier is given, for a new point $\vec{x}$ as

$$y(\vec{x})=\operatorname{sgn}\left(\sum_{i=1}^{n}\alpha_i\sum_{l,k=1}^{m}K_{l,k}x_{i,l}x_k+b\right), \tag{5}$$

with $K_{l,k}=\vec{x}_l\cdot\vec{x}_k$, and may be rewritten as

$$y(\vec{x})=\operatorname{sgn}\left(K(\vec{w},\vec{x})+b\right), \quad \vec{w}:=\sum_i \alpha_i\vec{x}_i. \tag{6}$$

Algorithm 1. Active learning against adversarial examples.

1: **procedure** Active learning against adversarial examples
2: **to find a point $\vec{x}$ that is in the $1-1/C$ quantile of 'informativeness' with probability $1-e^{-\beta}$, iterate $O(C\beta)$ times:**
3: uniformly at random sample a point $\vec{x}_{n+1}$ in the space.
4: evaluate, as explained in section 4.1, the probability that a point is in a given class c. We get $P_c(\vec{x}_{n+1})$.
5: solve the linear system of equation (9) using the procedure of [14], to obtain the state $|b,\vec{\alpha}\rangle_{n+1}$. It is explained in section 4.2.
6: perform (25), using the Chebyshev approach taken from [15], to calculate $|\vec{w}_{n+1}\rangle$. Then use amplitude estimation to estimate $|\vec{w}_{n+1}|$. The procedure to calculate $|\vec{w}_{n+1}\rangle$ is described in section 4.3, and its norm in appendix D.
7: calculate the informativeness $P_c(\vec{x}_{n+1})\cdot|\vec{w}_{n+1}|$ of point $\vec{x}_{n+1}$. If it is higher than the previous best 'informativeness', save the pair $(\vec{x}_{n+1}, P_c(\vec{x}_{n+1})\cdot|\vec{w}_{n+1}|)$ to memory substituting the previous best pair of values.
8: after the $O(\beta C)$ iterations, output the saved best pair.

Notice that the matrix $K_{l,k}$ is a kernel matrix that defines a dot product. Since the margins are of length at least 1, this means that for the training data

$$y_i(\vec{w}\cdot\vec{x}_i+b)\geqslant 1. \tag{7}$$

Since there are only two classes, $y_i=\pm1$, and so $y_i^2=1$. We will later say, with a bit of abuse of notation, that a given point is in class c to mean any of the two possible values of y_i. Additionally, we can transform the previous inequality results into equality adding slack variables e_i,

$$(\vec{w}\cdot\vec{x}_i+b)=y_i-y_ie_i, \tag{8}$$

such that if we allow $e_i>0$, we will be allowing for a soft margin, that is, some points may not fulfill (7). In such case we will add to Lagrangian (4) a term $(\gamma/2)\sum_j e_j^2$, where γ is specified by the user, to penalise any violation of (7). Then, minimising the Lagrangian is equivalent to solving the following least square approximation [14],

$$F\begin{pmatrix}b\\ \vec{\alpha}\end{pmatrix}=\begin{pmatrix}0 & \vec{1}^{\mathrm{T}}\\ \vec{1} & K+\gamma^{-1}\mathbf{1}\end{pmatrix}\begin{pmatrix}b\\ \vec{\alpha}\end{pmatrix}=\begin{pmatrix}0\\ \vec{y}\end{pmatrix}. \tag{9}$$

Here is where [14] uses quantum linear algebra techniques to solve (9), as we will see later, and obtain the result $|b,\vec{\alpha}\rangle$. To classify a new point $|\vec{x}\rangle$, one constructs, making use of qRAMs,

$$|\tilde{u}\rangle:=\frac{1}{\sqrt{N_{\tilde{u}}}}\left(b|0\rangle|0\rangle+\sum_{k=1}^{n}\alpha_k|\vec{x}_k||k\rangle|\vec{x}_k\rangle\right) \tag{10}$$

and

$$|\tilde{x}\rangle:=\frac{1}{\sqrt{N_{\tilde{x}}}}\left(|0\rangle|0\rangle+\sum_{k=1}^{n}|\vec{x}||k\rangle|\vec{x}\rangle\right). \tag{11}$$

Remember that qRAMs perform $\sum_k\beta_k|k\rangle\to\sum_k\beta_k|k\rangle|\vec{x}_k\rangle$, for β_k arbitrary amplitudes. Using the qRAM proposed in [13] only $O(\log n)$ gates are activated, although $O(n)$ should be present in the circuit.

If the previous is possible, one should also be able to prepare the state $|\psi\rangle=1/\sqrt{2}(|0\rangle|\tilde{u}\rangle+|1\rangle|\tilde{x}\rangle)$ and measures the probability of the ancilla being in state $|-\rangle=1/\sqrt{2}(|0\rangle-|1\rangle)$,

$$P=\frac{1}{2}(1-\langle\tilde{u}|\tilde{x}\rangle). \tag{12}$$

This is equivalent to performing a Hadamard gate over the first ancilla register and measuring the probability of obtaining $|1\rangle$, and is called swap test [31]. If $P\geqslant1/2$, $y(\vec{x})=1$, otherwise $y(\vec{x})=-1$.

4. Main algorithm

As we have seen, our aim is to sample points to be added to the training set, $\vec{x}_{n+1}$, with the highest possible informativeness, defined as $P_c(\vec{x}_{n+1})|\vec{w}_{n+1}|$, where the first term indicates the probability that a given point is in a class, and the second measures how much that would improve the classifier.

Thus, we will employ the strategy explained in the algorithm 1 to sample from the $1/C$ most relevant points.

4.1. Calculating $P_c(\vec{x}_{n+1})$

The first thing we should care about is calculating $P_c(\vec{x}_{n+1})$, the probability that point $\vec{x}_{n+1}$ is in class c.

The simplest way to calculate the probability would be to solve the SVM, calculate the distance from the point $\vec{x}_{n+1}$ to the decision boundary, and apply an activation function that converts the distance to a

probability. Solving the SVM can be done using [14] and reading each entry using amplitude estimation. It would return the vector $(b, \vec{\alpha})$, which can be used to create $\vec{w} = \sum_j \alpha_j \vec{x}_j$, and therefore the SVM.

A second, more elegant solution, is the following. In [14], the authors propose a method to estimate, using the swap test and the output state $|b, \vec{\alpha}\rangle$, to which class does a given point $|\vec{x}\rangle$ belong. They calculate that the success probability of measuring a $|-\rangle$ in the ancilla is

$$P = \frac{1}{2}(1 - \langle \tilde{u}|\tilde{x}\rangle), \tag{13}$$

for $|\tilde{u}\rangle$ and $|\tilde{x}\rangle$ defined as in (10) and (11). The interesting thing to notice is that if $P > 1/2$ the classification is in one class and if $P < 1/2$ it is in the other. We modify this protocol slightly so that we perform amplitude estimation on this result instead of repeatedly measuring the expected value, which improves the complexity from $O(\epsilon^{-2})$ to $O(\epsilon^{-1})$. This allows us to obtain the amplitude $A_{\vec{x}_{n+1}} = \sqrt{P}$. Then, one may use P as $P_c(\vec{x}_{n+1})$, or apply whatever activation function we consider appropriate to shape $P_c(\vec{x}_{n+1})$, like a sigmoid for example.

4.2. Quantum support vector machine

Now we focus on the main subroutine that our algorithm uses, which is mostly based in [14], and that outputs $|b, \vec{\alpha}\rangle$, the solution to (9). This subroutine will be key to calculating $|\vec{w}_{n+1}|$, as one can use (6) to calculate $\vec{w}$, whose norm we will estimate later. In this section we will review how [14] performs Hamiltonian simulation, and its role in the HHL algorithm [32], as well as some variations of that algorithm. We will also show the low rank approximation needed to achieve the exponential speedup, and the role of the condition number as a normalization factor in the estimation of $\|(b, \vec{\alpha})\|$. Finally, we will review the complexity of this calculation, which relies on the use of qRAMs to achieve an exponential speedup.

Thus, the first step is solving (9), which [14] does by using HHL algorithm [32] with a different Hamiltonian simulation. This is due to the fact that in general the kernel matrix K is dense, and the Hamiltonian simulation of [32] is better suited for the sparse case.

Rather, the authors of [14] use a technique they had previously developed in [33] that allows for efficient Hamiltonian simulation of dense low-rank matrices. Let us explain how to do it. Suppose we have a given quantum state σ, and we are able to prepare T copies of a density matrix ρ. One would like to perform the Hamiltonian simulation

$$e^{-it\rho}\sigma e^{it\rho}. \tag{14}$$

Repeated application of

$$\mathrm{tr}_1(e^{-i\Delta t S}\rho \otimes \sigma e^{i\Delta t S}) = \sigma - i\Delta t[\rho, \sigma] + O(\Delta t^2), \tag{15}$$

S the swap operator, approximates

$$e^{-iT\Delta t\rho}\sigma e^{iT\Delta t\rho}, \tag{16}$$

T indicating the number of times we apply (15). The authors of [33] observe that to simulate (14) with precision ϵ^{-1} one has to repeat the process $O(t^2\epsilon^{-1})$ times, each taking time $O(\log m)$ since that is the time it takes to prepare ρ using a qRAM.

In our case ρ is the matrix F, that will be represented by operator $\hat{F} = (J + K + \gamma^{-1}\mathbf{1})/\mathrm{tr}\ F$, where

$$J = \begin{pmatrix} 0 & \vec{1}^{\mathrm{T}} \\ \vec{1} & 0 \end{pmatrix}. \tag{17}$$

Then, $e^{i\Delta t\hat{F}} = e^{i\Delta tJ/\mathrm{tr}F} e^{i\Delta tK/\mathrm{tr}F} e^{i\Delta t\gamma^{-1}\mathbf{1}/\mathrm{tr}F}$. Simulating J is described in [34], and the matrix $\gamma^{-1}\mathbf{1}$ is also easy. An important insight of [14] is how to prepare a state with density matrix $K/\mathrm{tr}\ F$. If we were able to prepare $K/\mathrm{tr}\ K$ then one would only need to correct for a scalar term $\mathrm{tr}\ K/\mathrm{tr}\ F = O(1)$ in the simulation time. In the appendix A of [14] it is explained how to estimate $\mathrm{tr}\ K$, which can be used to estimate $\mathrm{tr}\ F = \mathrm{tr}\ K + \gamma \mathrm{tr}\ \mathbf{1}$.

So, let us show how [14] prepares a density matrix state $K/\mathrm{tr}\ K$. To do so, prepare, using a qRAM:

$$|\chi\rangle = \frac{1}{\sqrt{N_\chi}} \sum_i |\vec{x}_i||i\rangle|\vec{x}_i\rangle. \tag{18}$$

Tracing out the second register prepares

$$\mathrm{tr}_2(|\chi\rangle\langle\chi|) = \frac{1}{N_\chi} \sum_{i,j=1}^{n} \langle \vec{x}_i|\vec{x}_j\rangle |\vec{x}_i||\vec{x}_j||i\rangle\langle j| = K/\mathrm{tr}\ K. \tag{19}$$

The previous method allows to prepare the density matrix $K/\mathrm{tr}\,K$ needed to implement the Hamiltonian simulation of (15), in time complexity $O(\epsilon^{-1}t^2k)$. Therefore, we can perform the Hamiltonian simulation necessary to implement the HHL algorithm efficiently.

The HHL algorithm consists on the following steps. First, one formally decomposes the state $|0,\vec{y}\rangle$ of (9) in eigenvectors $|0,\vec{y}\rangle = \sum \beta_j|u_j\rangle$ of the Hamiltonian defined by matrix F in (9). Next, one phase estimates the eigenvalues, using a Hamiltonian simulation algorithm such as the one explained above, obtaining $\sum \beta_j|u_j\rangle|\lambda_j\rangle$. Finally, one performs the controlled rotations

$$\sum_j \beta_j|u_j\rangle|\lambda_j\rangle|0\rangle \to \sum_j \beta_j|u_j\rangle|\lambda_j\rangle\left(\frac{1}{\lambda_j\kappa}|1\rangle + \sqrt{1-\frac{1}{\lambda_j^2\kappa^2}}|0\rangle\right), \tag{20}$$

and postselects in ancilla state $|1\rangle$ to obtain the solution to the system of equations [32].

The stated complexity of the original HHL algorithm is $O(\kappa^2\epsilon^{-1}\mathrm{poly}\log n)$, κ the condition number. The complexity of the originally used Hamiltonian simulation algorithm is $O(d\epsilon^{-1}\mathrm{poly}\log n)$, where d the sparsity, for the sparse oracle access model [32]. The complexity of the condition number has been lowered to $O(\kappa)$, as shown using the technique of variable time amplitude estimation in [35], which is optimal in this parameter, also proved in that article. On the other hand, relying on more efficient Hamiltonian simulation techniques, [15] proposed a variation of the HHL algorithm that does not require amplitude estimation thus reducing the complexity from $O(\epsilon^{-1})$ to $O(\mathrm{poly}\log\epsilon^{-1})$. Unfortunately, the Hamiltonian simulation described above still has complexity $O(\epsilon^{\mp 1})$, making it impossible to reduce the complexity of the overall method further.

There is an alternative to the Hamiltonian simulation method that we have described. In [36], a technique is introduced that allows to simulate a dense Hamiltonian with complexity $O(\sqrt{n}\ \mathrm{poly}\ \log\ \epsilon^{-1})$, for Hamiltonians of any rank, and it can be used with the techniques from [15], to reduce the complexity on the precision from linear to polylogarithmic. The caveat is that it requires to use a special quantum-accessible data structure that is explained in the appendix B and plays the role of a quantum read-only memory, and has the complexity stated above, polynomial in n.

Coming back to our main discussion, once we have the state $|b,\vec{\alpha}\rangle$, we would like to recover the norm of vector $\vec{w}$. As a first step, we would like to obtain the norm $\|(b,\vec{\alpha})\|$. Suppose we are trying to solve $Ax = b$, where the largest eigenvalue of A is less or equal to 1; else see the end of appendix D for a minor correction. As explained in appendix A of [37], we can calculate the norm of the solution $\|x\|$ using

$$\|x\| = \kappa\|b\|\sqrt{p_1}, \tag{21}$$

where $\sqrt{p_1}$ represents the amplitude of the postselection ancilla of HHL algorithm being in the correct state, usually $|1\rangle$. This ancilla is used to perform the non-unitary part of the algorithm via a measurement. The reason for the previous equation (21) is because the acceptance probability of HHL scales as

$$p_1 = \frac{\|A^{-1}b\|}{\|b\|\kappa^2}. \tag{22}$$

Estimating such amplitude $\sqrt{p_1}$ thus requires of amplitude estimation [38], with cost $O(\epsilon^{-1})$.

Now let us turn to the condition number of the system of equations that appears in the SVM we are solving, the condition number of the matrix F in (9). Recall that the condition number is defined as

$$\kappa = \frac{\sigma_{\max}}{\sigma_{\min}}, \tag{23}$$

where $\sigma_{\min}$ and $\sigma_{\max}$ are the minimum and maximum singular values respectively. The condition number will be important to correct the norm of the solution to the system of equations, as can be seen from (21). However, calculating it with the quantum singular value estimation technique from [39] might be too expensive.

On the other hand, the authors of the quantum SVM article, [14], propose that in the case where the kernel matrix has $O(1)$ eigenvalues of size $O(1)$, and $O(n)$ eigenvalues with values $O(1/n)$ as it is in our case, we can choose a condition number $\kappa_{\mathrm{eff}} = O(1)$ such that in the end we will get an additional error of order $O(1/\sqrt{n})$, in addition to ϵ.

This low rank approximation means that the algorithm in [14] only takes into account the eigenvalues λ_i that are $\epsilon_K \leqslant \lambda_i \leqslant 1$. The main idea of the low rank approximation is filtering out those eigenvalues $\lambda < \kappa_{\text{eff}}^{-1}$, so the final rotation of the HHL algorithm, indicated in (20), is performed only if $\lambda_j > \kappa_{\text{eff}}^{-1}$ and imposes $\kappa = \kappa_{\text{eff}} = O(1)$.

In appendix C of the supplementary material of [14] the authors show that $\|K - K_{\text{eff}}\| = \sqrt{\sum_{\lambda_i = O(1/n)} \lambda_i^2}$, with K_{eff} the 'filtered' low-rank approximation of K, $K_{\text{eff}} = \sum_{\lambda_i \geqslant \kappa_{\text{eff}}^{-1}} \lambda_i |u_i\rangle\langle u_i|$. Since there are $O(n)$ eigenvalues of size $O(1/n)$, the induced error is of order $O(n^{-1/2})$. Then, one can use theorem 1 in the appendix of the HHL algorithm, [32], to show that, if $|b, \vec{\alpha}\rangle$ is the exact solution of the system of equation; and $|b, \vec{\alpha}\rangle_{\text{eff}}$ the approximate one after postselecting on the subspace spanned by the eigenvalues $\lambda_j \geqslant \kappa_{\text{eff}}^{-1}$ and the ancilla in state $|1\rangle$, then $\| |b, \vec{\alpha}\rangle_{\text{eff}} - |b, \vec{\alpha}\rangle \| = O(\epsilon + n^{-1/2})$. Thus, for relatively large n the induced error with this approximation is small.

Overall, this implies that instead of (21), we will have

$$\|x\| = \kappa_{\text{eff}} \|b\| \sqrt{p_1}. \tag{24}$$

In such case notice that we have imposed an effective condition number for all candidate points $\vec{x}_{n+1}$: $\kappa_{\vec{x}_{n+1}} = \kappa_{\text{eff}}$, so we no longer have to care about the condition number: it will be the same scale factor for all points $\vec{x}_{n+1}$, and thus without relevance. The same will happen also for $\|b\|$ in (24) since $b = (0, y_1, \ldots, y_{n+1})^{\text{T}}$, with y_{n+1} the same for all candidate points. Since we suppose n is large enough, the additional additive error we introduce will be small, of order $O(n^{-1/2})$ according to the appendix C in the supplementary material of [14]. Thus, the complexity of the algorithm is $O(\epsilon^{-3} \kappa_{\text{eff}}^3 \log(mn))$, where m is the dimension of the space.

To finish this section, let us recall how to calculate the complexity of the algorithm of [14]. They claim that (15) has error $\epsilon = \tilde{O}(\|\hat{F}\|^2 \Delta t^2)$, $\|\hat{F}\|$ the Frobenius norm of $\hat{F}$, and it is repeated T periods. Thus, $\epsilon = \tilde{O}(\|\hat{F}\|^2 \Delta t^2 T) = \tilde{O}(\|\hat{F}\|^2 t^2 / T)$, for $\Delta t = t/T$. This implies a simulation cost $T = \tilde{O}(t^2 \|\hat{F}\|^2 \epsilon^{-1})$, when implementing (15) is taken at unit cost.

On the other hand, HHL requires phase estimating the eigenvalues, so the relative error of λ^{-1} can be indicated as $\epsilon = O(1/\lambda t) \leqslant O(\kappa_{\text{eff}}/t)$. Therefore $t = O(\kappa_{\text{eff}} \epsilon^{-1})$, and substituting in the previous paragraph, $T = O(\|\hat{F}\|^2 \kappa_{\text{eff}}^2 \epsilon^{-3} \log(mn))$. An additional κ_{eff} is needed to perform postselection on the result. Thus, the overall complexity of the algorithm is $O(o \|\hat{F}\|^2 \kappa_{\text{eff}}^3 \epsilon^{-3} \log(mn))$, for a kernel of order o.

If instead of using the basic HHL technique we had decided to use more advanced ones, such as the Fourier approach described in [15] (see appendix B), the complexity would have been $\tilde{O}(T\alpha)$, for $T = \tilde{O}(\|\hat{F}\|^2 t^2 \epsilon^{-1})$, $t = \tilde{O}(\kappa)$ and $\alpha = \tilde{O}(\kappa)$, ignoring polylogarithmic factors [15]. Overall, the complexity would be $\tilde{O}(\|\hat{F}\|^2 \kappa^3 \epsilon^{-1})$.

4.3. The norm of $|\vec{w}_{n+1}|$

The aim of the previous section was to obtain $|b, \vec{\alpha}\rangle$ and its norm. The aim of this one is to calculate $\vec{w}$ from that result. We will see that one may do that using a Fourier expansion, at cost $O(\epsilon^{-1})$; or better, use of Chebyshev series as in [15], at cost polylogarithmic in all variables, provided the use of qRAMs. In appendix D we explain how to calculate the norm of $|\vec{w}_{n+1}|$ from $\vec{w}$, which only requires performing amplitude estimation once, and normalizing with some factors.

The first, trivial, idea we had to calculate $|\vec{w}_{n+1}\rangle$ is reading all the entries of $|b, \vec{\alpha}\rangle$ using amplitude estimation. Then one may use classical computing to compute $\vec{w} = \sum_{i=1}^{n+1} \alpha_i \vec{x}_i$, and finally its norm $|\vec{w}_{n+1}| = \sqrt{\sum_{i=1}^{m} (\sum_{j=1}^{n+1} x_{ij, x_{n+1}} \alpha_{j, x_{n+1}})^2}$. Amplitude estimation does not immediately recover the sign of each α_i, but finding it is not complicated either. Once we know $|\alpha_i|$ and $|\alpha_j|$ we can prepare the state $C_{ij}(|\alpha_i||j\rangle + |\alpha_j||i\rangle)$ and perform a swap test with the solution vector $|b, \vec{\alpha}\rangle$. If the relative sign of entries i and j is the same we will get a nonzero result proportional to $2C_{ij}\alpha_i\alpha_j$, but if the relative sign is opposite, the dot product will cancel out [40]. Establishing the relative sign of two entries is enough, since the norm will not care about the global sign.

However, the procedure described above is time consuming, as one needs to prepare the solution of the system of equation (9) $O(n+1)$ times in order to read all the entries of the solution and calculate their relative sign. Also, calculating $\vec{w}$ takes complexity $O(nm)$ using classical computing, so, instead of that we will try to prepare $|\vec{w}_{n+1}\rangle$ from $|b, \vec{\alpha}\rangle$, and estimate the change in the norm of the result. Notice in the first place that the quantum SVM article [14] is able to classify a point without first calculating $|\vec{w}_{n+1}\rangle$, as indicated in (12) and subsequent paragraph. Also, it is not clear how to prepare $|\vec{w}_{n+1}\rangle$ from the state in (10).

Instead, since $\vec{w} = \sum_i \alpha_i \vec{x}_i$, this suggest multiplying the solution vector from the quantum SVM, $|b, \vec{\alpha}\rangle$ by a matrix operator whose entries are $x_{i,j}$. Explicitly,

$$A|b,\vec{\alpha}\rangle = \begin{pmatrix} 0 & x_{1,1} & \dots & x_{1,m} \\ \vdots & & & \vdots \\ 0 & x_{n+1,1} & \dots & x_{n+1,m} \end{pmatrix} \begin{pmatrix} b \\ \alpha_1 \\ \vdots \\ \alpha_{n+1} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{m} \alpha_j x_{1,j} \\ \vdots \\ \sum_{i=1}^{m} \alpha_j x_{n+1,j} \end{pmatrix} = \vec{w}. \tag{25}$$

Performing this operation classically, requires $O(nm)$ operations. Can we perform this operation in a quantum way?

Our strategy to calculate the previous matrix-vector product will take inspiration from the algorithms presented in [15] to simulate A^{-1}. The fact that in our case A is not necessarily square should not pose any problem, since one can write

$$\begin{matrix} m+1 \\ n+1 \end{matrix} \begin{pmatrix} m+1 & n+1 \\ \mathbf{0} & A^{\mathrm{T}} \\ A & \mathbf{0} \end{pmatrix} \begin{pmatrix} b \\ \vec{\alpha} \\ \vec{0}_{n+1} \end{pmatrix} = \begin{pmatrix} \vec{0}_{m+1} \\ \vec{w} \end{pmatrix}. \tag{26}$$

Let us call M the matrix from this previous equation. The first thing we have to do is to attach a register to $|b, \vec{\alpha}\rangle$, that we set to $|0\rangle$ for the entry b (that is, the original register is in state $|0\rangle$) and $|1\rangle$ otherwise. This will allow us to substitute any of the 0s in the matrix in (25), but avoiding interference of the entry of b with those of $\vec{\alpha}$. The aim of this is to avoid M being ill-conditioned, although it is only a technical detail. Then all the singular values fall in $[1/\kappa_M, 1]$, and we can rewrite

$$\begin{pmatrix} b \\ \vec{\alpha} \\ \vec{0}_{n+1} \end{pmatrix} \to \begin{pmatrix} b \\ \vec{0}_m \\ \vec{0}_{n+1} \end{pmatrix} \otimes |0\rangle + \begin{pmatrix} 0 \\ \vec{\alpha} \\ \vec{0}_{n+1} \end{pmatrix} \otimes |1\rangle. \tag{27}$$

Notice that by linearity of quantum mechanics the unitary we were going to apply to the the left-hand side is applied individually to each of the two terms in the decomposition. Later we will only care about the case when the additional ancilla we have added is in state $|1\rangle$.

With this remark, the question becomes twofold: how to decompose M into a linear combination of unitaries (LCU), as it is done with A^{-1} in [15], and how to simulate those linear operators efficiently. Notice that now $\mathrm{e}^{-\mathrm{i}M}$ is a unitary operator.

The first question is how to decompose M in a linear combination of unitaries. If we choose, as it is frequently done, unitary operators of the form $\mathrm{e}^{-\mathrm{i}Mt}$, we can write $M = \sum_i \beta_i \mathrm{e}^{-\mathrm{i}Mt_i}$. Then, as both sides of the equality are diagonal in the same basis, one can consider this is equivalent to writing $x = \sum_i \beta_i \mathrm{e}^{-\mathrm{i}xt_i}$. The first idea is then to decompose x as a Fourier series. Specifically, decomposing x gives us

$$x = \sum_{t=1}^{\infty} \frac{(-1)^t}{t} 2\,\sin(tx) = i\sum_{t=1}^{\infty} \frac{(-1)^t}{t}(\mathrm{e}^{-\mathrm{i}tx} - \mathrm{e}^{\mathrm{i}tx}). \tag{28}$$

Since we cannot perform the entire summation, the question is how to truncate the series maintaining a given precision ϵ^{-1}. Call $S_{\bar{t}}(x)$ the truncated series up to $\bar{t}$. Since all the singular values will be in $[1/\kappa_M, 1]$, with κ_M the condition number of M, then, we want to study the series in the range $[0, 1]$. Notice that $f(x) = x$ is Lipschitz-continuous with constant 1, so can use a result from [41] to bound the error incur when truncating the series. According to corollary 1 in [41], we can say that $|f(x) - S_{\bar{t}}(x)| \leqslant (a\ \log\ \bar{t})/\bar{t}$, where a is a constant. Thus, if we want $|x - S_{\bar{t}}(x)| \leqslant \epsilon$, it is enough to choose $\bar{t}/\log\ \bar{t} = O(\epsilon^{-1})$. So the complexity is almost linear in ϵ^{-1}, as we have to Hamiltonian-simulate M during time $\bar{t}$. Furthermore, if we chose a Hamiltonian simulation algorithm, as M is dense, the cost will be relatively large, $\tilde{O}(\sqrt{n+m})$ if we were to use [36] for instance. In any case we would not obtain an exponential advantage, since Hamiltonian simulation of dense Hamiltonians is not expected to have complexity $\log n$ in general [42]. Also notice that we cannot use the trick from the previous section, since that requires M to be a density matrix and $\mathrm{tr}\, M = 0 \Rightarrow M$ is not a density matrix.

A second, better approach, also inspired by [15], is to decompose the matrix in Chebyshev polynomials, $x = \sum \gamma_i \mathcal{T}_i(x)$. In fact, this decomposition in polynomials is quite simple since the first Chebyshev polynomial is

$$\mathcal{T}_1(x) = x, \tag{29}$$

so the expansion will have a single term and will be exact, which is very positive. Let us now look at the cost of simulating a Chebyshev polynomial for a $\bar{n} \times \bar{n}$ d-sparse matrix M. One does that via quantum walks. Define a quantum walk as set of states $\{|\psi_j\rangle \in \mathbb{C}^{2\bar{n}} \otimes \mathbb{C}^{2\bar{n}};\ j \in [\bar{n}]\}$ [15]:

$$|\psi_j\rangle := |j\rangle \otimes \frac{1}{d} \sum_{k\in[\bar{n}];A_{jk}\neq 0} \left(\sqrt{A^*_{jk}}|k\rangle + \sqrt{1-|A_{jk}|}|k+\bar{n}\rangle \right). \tag{30}$$

Then, one defines the isometry

$$T = \sum_{j\in[\bar{n}]} |\psi_j\rangle\langle j|, \tag{31}$$

and S the swap operator $S : |j,k\rangle \to |k,j\rangle$. The walk operator is $W := S(2TT^\dagger - \mathbf{1})$. With these definitions, and using lemma 15 of [15] enunciated in the appendix B, one can prove that for any state $|\psi\rangle$, substituting T by its corresponding unitary T_U [15]

$$T_U^\dagger W T_U |0^{\lceil \log 2\bar{n}\rceil+1}\rangle|\psi\rangle = |0^{\lceil \log 2\bar{n}\rceil+1}\rangle \mathcal{T}_1(\overline{M})|\psi\rangle + |\Phi^\perp\rangle = |0^{\lceil \log 2\bar{n}\rceil+1}\rangle \overline{M}|\psi\rangle + |\Phi^\perp\rangle, \tag{32}$$

for $\overline{M} = M/d$, and $|\Phi^\perp\rangle$ such that $(|0^{\lceil \log 2\bar{n}\rceil+1}\rangle\langle 0^{\lceil \log 2\bar{n}\rceil+1}| \otimes \mathbf{1})|\Phi^\perp\rangle = 0$.

Let us first study the cost of applying T. According to lemma 10 in [43], to implement T one must apply $\log d$ Hadamard gates, followed by $O(1)$ calls to an oracle O_F that outputs the column index of the lth non-zero element of a row j of the matrix, $O_F : |j,l\rangle \to |j, f(l,j)\rangle$; and the sparse-access oracle O_M, that outputs entry M_{jk} on input (j,k). For simplicity and without loss of generality let us assume that in (26) all entries of A are non-zero. If such is the case, then the cost of the oracles is not high either. The first oracle outputs $O_F : |j,l\rangle \to |j,l\rangle$ or $O_F : |j,l\rangle \to |j,l+n+1\rangle$ depending on whether we are on the last $n+1$ rows or in the first $m+1$, respectively. The second oracle performs $O_M : |j,k\rangle|z\rangle \to |j,k\rangle|z \oplus M_{jk}\rangle$, which is nevertheless no more restrictive than the qRAM model that we were using to solve the quantum SVM.

What is the cost of implementing W then? In the proof of their theorem 4, [15] cites lemma 10 from [43] to explicitly state that if we want to simulate W with error ϵ', its complexity and the complexity of (32) is $O(\log \bar{n} + \log^{2.5}(\kappa d/\epsilon'))$. This is not surprising since the cost only depends on T, its inverse $T^\dagger$, and a swap gate. Notice that our matrix is dense, so $d = O(\bar{n}) = O(n+m)$, but still the cost of this process is polylogarithmic in all variables! In fact this method seems applicable to any matrix M, sparse or not, whose entries are accessible through a sparse-access oracle in the form of a qRAM. One may even say that the qRAM is hiding the cost, since it requires time to prepare the data and also has $O(n)$ quantum gates even if it only uses a small number of them. Notice also that the linear dependence on d and κ for the quantum linear system algorithm presented in [15] does not come from implementing W but because one has to apply it $O(d\kappa)$ times in the series expansion of A^{-1}. In our case we only have to apply W once, so we do not have such linear complexity term.

Finally, we measure the amplitude of $|1\rangle$ of the ancilla that we attached to the state to mark the entry of b after (25). The amplitude of such state is, after taking into account a normalization factor for M and $\|(b,\vec{\alpha})\|$, precisely $|\vec{w}_{n+1}| = \sqrt{\sum_{i=1}^{m} (\sum_{j=1}^{n+1} x_{ij,x_{n+1}} \alpha_{j,x_{n+1}})^2}$, what we were looking for. We can measure that amplitude using amplitude estimation.

In appendix D we explain in greater detail how to correct the value extracted with amplitude estimation, to find the norm $|\vec{w}_{n+1}|$ provided either the linear combination of unitaries with the Fourier series, or the Chebyshev polynomial approach. The cost of amplitude estimation is $O(\epsilon^{-1})$. Having calculated $|P_c(\vec{x}_{n+1})|$ and $|\vec{w}_{n+1}|$, the informativeness of a given point is just the product of both.

4.4. Finding the target point $\vec{x}_{n+1}$

Now that we know how to calculate the 'informativeness' of a given point, let us explain why the algorithm 1 finds, with probability $1 - e^{-\beta}$ a point that is in the quantile $1 - 1/C$ of informativeness. The arguments presented here are the same to those in appendix C.

If we sample uniformly at random, and calculate the informativeness of c points, the probability that none of those points is in the quantile $1 - 1/C$ is clearly

$$p = \left(1 - \frac{1}{C}\right)^c. \tag{33}$$

We want to make such probability $p < e^{-\beta}$, so that at least one is in the $1 - 1/C$ quantile. That means

$$c > \frac{-\beta}{\log\left(1 - 1\frac{1}{C}\right)}. \tag{34}$$

Since

$$\lim_{C\to\infty} \frac{\frac{-1}{\log\left(1-\frac{1}{C}\right)}}{C} = 1, \tag{35}$$

it is clear then that the number of sampled points should be $c = O(\beta C)$. This explains why our procedure is efficient in these variables. More information on alternative strategies can be found in the appendix C.

5. Main results

In this article we propose a theoretical framework that allows us to think of active learning as sampling the most promising new points to be classified, so that the minimum distance between classes can be found, and theorem 1 used. We also propose a quantum algorithm that would allow us to perform the sampling efficiently with an exponential advantage over what is achieved classically.

In the quantum algorithm we do not use neural networks but rather an SVM [44], although it might be possible to use the general strategy in other setups using quantum neural networks.

Let us explain further the complexity comparison with the classical algorithms. One may argue that a good quantum strategy to solve problem 1 would be to use amplitude estimation and bisection search to find a threshold for the quantile $1 - 1/C$, for example the 1% most relevant points, in which case $C = 100$. Then, one may use amplitude amplification to find one of the points in such quantile. This achieves an exponential advantage over the classical case when one wants to find such points using classical computing with certainty. In the usual case of amplitude amplification and amplitude estimation we have an oracle that tells us whether an element is marked or not, and this yields a quadratic advantage with respect to the classical case. However, in our situation, being a good point or not depends on its relative 'informativeness' with respect to other points. This means that classically, in order to assert with certainty that a given point is within the top $1/C = 1\%$ quantile, one should first calculate the informativeness of $0.99N$ points, out of N points. This is clearly prohibitive, since $N = O(l^m)$, where m is the dimension of the space, and l its discretisation. Notice for example that for a $n_0 \times n_0$ image, the dimension of the image would be $m = 3n_0^2$, due to the three colours or channels needed to define each pixel.

In contrast, if we want to solve this problem using amplitude estimation we do not incur in such cost. What we do is find, using bisection and amplitude estimation, an informativeness threshold above which there are only $1/C$ of the most informative points. Once that is the case, we can mark those points and use amplitude amplification to find them [38]. The cost will then be $O(C\epsilon^{-1}\beta)$ for a given precision ϵ in the threshold, and success probability $1 - e^{-\beta}$, and crucially C independent of N. Amplitude amplification also bears a cost $O(\sqrt{C})$.

On the other hand, one can also find probabilistic classical strategies that output a point that is in the $1 - 1/C$ quantile with probability exponentially high, $1 - e^{-\beta}$, and the complexity would be polynomial in parameters C and β. Since our quantum strategy would also have some exponentially small probability of failure, and the complexity would also be polynomial in those parameters, the advantage would be unclear. We discuss this point in depth in the appendix C, for a problem that generalises our own, and find that this advantage would be quadratic in some parameters.

However, the algorithm that we present here achieves an exponential advantage over its classical counterpart. The speedup is due to a faster calculation of the 'informativeness' of a given point, thanks to the use of qRAMs, the algorithm for quantum SVM [14], and an approach to perform matrix-vector products using Chebyshev polynomials taking advantage of techniques developed in [15, 43]; in conclusion, a better use of quantum linear algebra techniques. Our algorithm here has overall complexity $\tilde{O}(C\epsilon^{-1}\,\beta(o\|F\|^2\kappa_{\text{eff}}^{-3}\epsilon^{-3} + \text{poly}\log(\kappa_M\epsilon^{-1}(n+m))))$, where $\|F\|$ is the Frobenius norm of the matrix in (9), o the order of the kernel, $\kappa_{\text{eff}} = O(1)$ an effective condition number that we choose, ϵ the precision on the 'informativeness' of a given point, $1 - e^{-\beta}$ the success probability with which one wants to be sure that the chosen point is in the $1 - 1/C$ quantile, and κ_M the condition number of matrix M appearing in (26).

In contrast, a classical algorithm would have polynomial complexity in n and m due for instance to matrix-vector product (26), or the final calculation of the norm of $|\vec{w}|$, central to our discussion.

In the next subsection we lay out the general strategy of our paper to solve this problem.

6. Complexity

In this section we want to give a calculation of the complexity of the proposed quantum algorithm.

The first step is calculating the probability that a given point of the high dimensional space is in a certain class, $P_c(\vec{x}_{n+1})$. This implies solving a quantum SVM [14], with complexity $O(o\|F\|^2\epsilon^{-3}\kappa_{\text{eff}}^{-3}\log(mn))$, where o is the order of the kernel, $\|F\|$ is the Frobenius norm of the matrix that appears in (9), m the dimension of the space, n the number of already classified points, and κ_{eff} is an effective condition number which can be taken $O(1)$ for this problem.

Reading out the probability using amplitude estimation in the swap test costs an additional multiplicative $O(\epsilon^{-1})$, so the overall complexity of this step is $O(o\|F\|^2\epsilon^{-4}\kappa_{\text{eff}}^{-3}\log(mn))$. Since we are assuming that we have scaled the matrix to obtain the largest eigenvalue equal to 1, the procedure of [14] implies that the smallest one that we take into consideration is $\lambda^{-1} = \kappa_{\text{eff}}$. The low-rank approximation induces an additional error of $O(n^{-1/2})$, as it is explained in the supplementary material of the original article [14].

Next step is calculating the quantum SVM for the system with the added point, again with complexity $O(o\|F\|^2\epsilon^{-3}\kappa_{\text{eff}}^{-3}\log(mn))$. Then, preparing $|\vec{w}_{n+1}\rangle$ has complexity $O(\text{poly}\log(\kappa_M\epsilon^{-1}(n+m)))$, as reported in the corresponding section. Calculating $|\vec{w}_{n+1}|$ also have an additional multiplicative complexity of $\tilde{O}(\epsilon^{-1})$ due to the amplitude estimation, given the procedure in appendix D.

Finally, if we are interested in finding a point in the $1-1/C$ quantile of 'informativeness' with probability $1-\mathrm{e}^{-\beta}$, then the procedure explained in section 4.4 implies iterating the procedure over $O(C\beta)$ points and selecting the best.

Thus, in general, the complexity will be $\tilde{O}(C\beta\epsilon^{-1}(o\|F\|^2\kappa_{\text{eff}}^{-3}\epsilon^{-3} + \text{poly}\log(\kappa_M\epsilon^{-1}(n+m))))$, with error $\tilde{O}(\epsilon + n^{-1/2})$. The $n^{-1/2}$ comes from the low-rank approximation of [14], as it is explained on its appendix C.

7. Conclusion

In the previous sections we have seen that provided the use of qRAMs, it could be possible to establish a polynomial-complexity sampling procedure which lets us know what are the most promising points to be added to the training set in order to get a better approximation of the minimum distance between classes, r_p. Recall that knowing r_p is a requisite to applying a δ-cover and, using theorem 1, avoid adversarial examples. This protocol is heuristic, which means that in order to check its actual performance we have to run it in a realistic quantum computer.

In this paper we have presented a procedure that allows to solve this problem efficiently, in time polylogarithmic in the dimension of the space m and number of already classified points, n; and polynomial in C, β and the precision ϵ^{-1} of the informativeness estimate.

There are nevertheless two important shortcomings of our algorithm. The first is that, since we rely heavily on reference [14], and they use qRAMs to achieve their speedup, we also need qRAMs. The second is that there are some minor parts of the general algorithm that will necessarily have polynomial complexity in the dimension of the space m. Those are: sampling initial points, and loading their value in the qRAM. We strongly believe that this should nevertheless be no problem. The sampling procedure for instance will hardly be inefficient and can be easily done while the previous point is being processed. The loading in the qRAM is perhaps more expensive, but may be done while the point is asked to be classified by a human. Therefore we believe that the complexity result is still very valid in practice.

Finally, we want to recall that our work relies heavily on reference [14]. At the time that article was written, there was no known method to solve low-rank linear systems of equations in polylogarithmic time. However, [45] has recently proposed a method that achieves precisely that, complexity $O(\|F\|^6k^6\kappa^{16}\epsilon^{-6})$, k the rank. It relies on the techniques developed by Ewin Tang [46], which found a classical algorithm taking inspiration from [47]. Therefore, one could also solve the linear system of equations that appear in this case in polylogarithmic time in the dimension n. We have two arguments why we have not explored the result here. Firstly, although [45] is efficient on n, it has a pretty bad behaviour in other involved parameters such the error ϵ. Secondly, perhaps more important argument, is that even though the most complicated part of the algorithm, the solution to (9), can be performed with this quantum-inspired algorithm in polylogarithmic complexity; solving (26), calculating the norm of $|\vec{w}_{n+1}|$, and performing the inner product needed to calculate $P_c(\vec{x})$, are, to the best knowledge of the authors of this article, not possible in polylogarithmic time. Thus, even though we think the exponential advantage could perhaps be reduced to a 'mere' polynomial one using classical computing, we do not think there is an explicit method to do it, but only a line of research that may do it in the future.

In conclusion, in this article we have framed a possible solution to adversarial examples using active learning, and a sampling methodology to find the most important points that could be added to the training set. We have also presented an algorithm that provides an exponential advantage over its classical counterpart, provided the use of qRAMs.

Acknowledgments

We would like to thank Santiago Varona for useful comments on the manuscript, as well to Jaime Sevilla, Nikolas Bernaola and Javier Prieto for pointing us to useful statistic results for appendix C. We acknowledge financial support from the Spanish MINECO grants MINECO/FEDER Projects FIS 2017-91460-EXP, PGC2018-099169-B-I00 FIS-2018 and from CAM/FEDER Project No. S2018/TCS-4342 (QUITEMAD-CM). The research of MAM-D has been partially supported by the US Army Research Office through Grant No. W911NF-14-1-0103. PAMC thanks the support of an FPU MECD Grant.

Appendix A. A provably robust classifier

In this appendix we would like to review the theorem of [12] that is the basis for the provably robust classifier. We introduce informal definitions for a δ-*cover* of a manifold, the ϵ_0-*neighbour* of a submanifold, and finally what is an *adversarial example*. A formal definition for those concepts can be seen at appendix B.

Informally stated, a δ-*cover* of a manifold is a set of points $\{\vec{x}_i\}$ of the manifold such that the set of balls with centers $\{\vec{x}_i\}$ and radious δ contains the manifold. Or in other words, any point at the manifold is at most δ-far from a point from $\{\vec{x}_i\}$. Relatedly, an ϵ_0-*neighbour* of a given submanifold is composed of all those points in the manifold at most ϵ_0-far from the submanifold. In both cases we are assuming a p-norm.

An example of δ-cover is depicted in figure A1 in green and will be a key ingredient to avoid adversarial examples. With perhaps an abuse of notation, we will call a δ-cover simultaneously to the set of $\vec{x}_i$ points that are in the center of these balls, and to the balls themselves. This means that for δ the parameter that controls how coarse or fine is the sampling, a δ-cover is a coarse-grained sampling of each class. Following our previous example, an example of a δ-cover is a set of images of cars, for example, such that any possible image of a car is no further than δ-far to one of the training set. Notice that one can measure distance between images by the distance between the vectors containing the amount of green, blue and red of each pixel, in a p-norm.

For the next definition we will need the notion of a *classifier*, a function f that, given a point x is able to predict a label y. An ϵ_0-neighbour is also depicted in figure A1 in dashed lines around the red and blue submanifolds. Here, ϵ_0 has the meaning of the robustness against perturbations. For example, take an image of a car and its corresponding vector. ϵ_0 is the amount one can perturb the vector without fooling the classifier. Then, $\mathcal{M}^{\epsilon_0}$ is the space of such perturbations of size ϵ_0, for each class within submanifold $\mathcal{M}$, that contains the different classes.

Provided the previous definitions, an ϵ_0-*adversarial example* can be described as a point in the manifold that, being ϵ_0-close to a class or submanifold, is mistakenly classified as an example of another class. For such definition to make sense ϵ_0 must be smaller than the distance between two classes r_p. Else, a given point in the region $\mathcal{M}^{\epsilon_0}$ could be ϵ_0-close to two classes at the same time. An example of an adversarial example is pictured in figure A1. The blue point is ϵ_0-close to the red class but classified as blue.

The intuition of [12] to avoid ϵ_0-adversarial examples is to cover all classes with a δ-cover, such that any point in the ϵ_0-neighbour of the classes is δ-close to a correctly classified point. What we want to find out is how big can δ be in order to maintain protection against adversarial examples. Such δ will depend crucially on the minimum distance of separation between classes, $\delta < r_p - \epsilon_0$, and thus our main objective in this article is to sample efficiently in an active learning setting, to find this minimum distance r_p. The precise statement of the theorem can be found in appendix B, but intuitively we are trying to cover the space of $\mathcal{M}^{\epsilon_0}$ corresponding to each class, with sets of balls, such that the sets do not overlap but the balls are as large as possible to avoid sampling more than it is needed.

Therefore, if we knew the minimum distance of separation of two classes, r_p, we would be able to produce a cover of the two classes that avoids the adversarial examples. However, finding r_p is not easy, because we only have an upper bound to the minimum distance of samples between two classes. The generalisation to a small arbitrary number of classes is usually done by pairs. The problem we aim to solve in this paper is finding this minimum distance between the two classes r_p, because if we overestimate δ we would not be able to use theorem 1, and if we underestimate it, the δ-cover would be more expensive to establish.

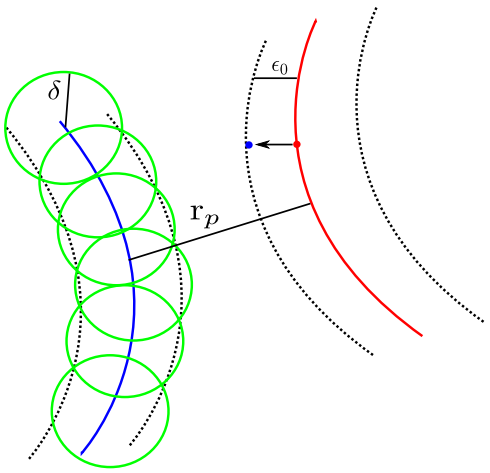

Figure A1. Submanifold $\mathcal{M}$ made of two classes separated by a distance r_p. The space near each class is $\mathcal{M}^{\epsilon_0}$. We also represent a δ-cover of the blue class. The red point is in the red class, but when ϵ_0-shifted, can be mistakenly classified as blue. It would be an adversarial example that we are trying to avoid.

Given these definitions and the previous theorem, we can think of this as a procedure to establish an ϵ_0-tolerance to perturbations. The robustness provided by the δ-cover qualitatively means that, if we provide a cover of the class with balls of size δ, then any point ϵ_0-near the class, in $\mathcal{M}^{\epsilon_0}$, will be correctly classified. Thus, our classifier will be robust to perturbations. Therefore, the level of robustness ϵ_0 is something we choose, and r_p is the unknown we are looking for that would allow us to calculate δ.

Appendix B. Technical definitions and previous results

In the main text we gave informal definitions about concepts needed to understand theorem 1. Here we give them rigorously. The first concept we need to introduce is that of a δ-cover of a manifold, which will be used in that theorem.

Definition 3. Given a manifold $\mathcal{M}$, a δ-cover of such manifold is a set of balls of radius δ, $\bigcup_{i\in I} B(x_i, \delta)$, such that $\forall x \in \mathcal{M}\ \exists j \in I | x \in B(x_i, \delta)$.

Additionally, in order to understand the meaning of an adversarial example, we need to define an ϵ-neighbour:

Definition 4. Given a manifold $\mathcal{M} \subset \mathbb{R}^m$, an ϵ-neighbour of $\mathcal{M}$ in the norm p, $\mathcal{M}^\epsilon$, is the set of points $x \in \mathbb{R}^m$ such that the p-distance of x to $\mathcal{M}$, $d_p(x, \mathcal{M}) \leqslant \epsilon$.

Finally, an adversarial example is defined as

Definition 5. Let $\mathcal{M} \subset \mathbb{R}^m$ be a manifold containing several disjoint parts called classes C_i separated by a distance $r_p \gg \epsilon$ in the norm p, and a classifier f reasonably well trained to distinguish between those classes. An ϵ-adversarial example of such classifier in the norm p is a point x whose p-distance to a given class C_0 is smaller than ϵ, but it is classified to be in class C_1. That is $d(x, C_0) \leqslant \epsilon$ and $f(x) = C_1$.

With all the previous definitions, we can state the theorem 1 that guarantees resistance to adversarial examples.

Theorem 1 (Khoury and Hadfield-Menell) [12]. *Let $\mathcal{M} \subset \mathbb{R}^m$ be a k-dimensional manifold that contains each of the classes, and let r_p be the minimum separation distance between two classes in norm p, $r_p > \epsilon$. Let $\mathcal{L}$ be a learning algorithm, and $f_\mathcal{L}$ the classifier it produces. Assume that for any point x in the training set $X_\mathcal{L}$ with label y, and any point $\hat{x} \in B(x, r_p)$, the learning algorithm classifies $f_\mathcal{L}(\hat{x}) = f_\mathcal{L}(x) = y$.*

We then have the following guarantee: if $X_\mathcal{L}$ is a δ-cover for $\delta < r_p - \epsilon$ then $f_\mathcal{L}$ correctly classifies $\mathcal{M}^\epsilon$, that is, an ϵ-neighbour of $\mathcal{M}$.

In this appendix we also give an overview of the quantum-accessible data structure that we use to perform the Hamiltonian simulation of dense matrices.

Theorem 2. [39, 48]: *let $M \in \mathbb{R}^{n'\times n'}$ be a matrix. Let w be the number of nonzero entries. Then there exists a quantum-accessible data structure of size $O(w \log^2(n'^2))$, which takes time $O(\log(n'^2))$ to store or update a*

single entry. Using this data structure, there is a quantum algorithm able to perform the following maps with error ϵ:

$$U_{\mathcal{M}} : |i\rangle|0\rangle \to \frac{1}{\|M_{i\cdot}\|}\sum_j M_{ij}|ij\rangle; \tag{B1}$$

$$U_{\mathcal{N}} : |0\rangle|j\rangle \to \frac{1}{\|M\|_F}\sum_i \|M_{i\cdot}\||ij\rangle; \tag{B2}$$

where $\|M_i\|$ is the l_2-norm of row i of M. The complexity of this algorithm is $O(\text{poly}\log(n'/\epsilon))$. This means in particular that given a vector f in this data structure, we can prepare an ϵ approximation of it, $1/\|v\|_2\sum_i v_i|i\rangle$, in time $O(\text{poly}\log(n'/\epsilon))$. □

The previous data structure can be used for dense Hamiltonian simulation [36], into the main algorithm of [15]

Theorem 3. [36] **Hamiltonian simulation of dense Hamiltonian**. *Let H be a 2^n-dimensional Hamiltonian stored in the quantum-accessible data structure. Then there is a quantum algorithm able to simulate it in complexity*

$$O\left(t\Lambda n\,\log^{5/2}(t\Lambda\epsilon^{-1})\frac{\log(t\|H\|\epsilon^{-1})}{\log\,\log(t\|H\|\epsilon^{-1})}\right), \tag{B3}$$

for t the simulation time, ϵ^{-1} the precision and $\Lambda = \max\{\|H\|, \|H\|_1\}$, taking into account that $\|H\|_1 \leqslant \sqrt{2^n}\|H\|$, but taking $\|H\|$, the spectral norm, as measure of cost.

The Hamiltonian simulation is very useful to solve linear systems of equations. The first and most famous of those algorithms is usually called HHL after their discoverers Harrow, Hassidim and Lloyd.

Theorem 4. [32] **(HHL)**. *Let M be an $n' \times n'$ Hermitian matrix (if the matrix is not Hermitian it can be included as a submatrix of a Hermitian one) with condition number κ and M having an sparsity d (at most d nonzero entries in each row).*

Let b be an n'-dimensional unit vector, and assume that there is an oracle $\mathcal{P}_b$ which produces the state $|b\rangle$, and another $\mathcal{P}_M$ which, taking (r, i) as input, outputs the location and value of the i'th nonzero entry in row r of M. Let

$$x = M^{-1}\,b, \quad |x\rangle = \frac{x}{\|x\|}. \tag{B4}$$

Then, there is an algorithm that outputs $|x\rangle$ in complexity $O(d\kappa^2\epsilon^{-1}\text{poly}\log n')$. □

However, since it was published, more efficient methods have been published. In particular, [15, 35] achieved improvements in the condition number and precision respectively. We present here the main result from the later.

Theorem 5. [15] *Let M be an $n' \times n'$ Hermitian matrix (if the matrix is not Hermitian it can be included as a submatrix of a Hermitian one) with condition number κ and M having an sparsity d (at most d nonzero entries in each row).*

Let b be an n'-dimensional unit vector, and assume that there is an oracle $\mathcal{P}_b$ which produces the state $|b\rangle$, and another $\mathcal{P}_M$ which, taking (r, i) as input, outputs the location and value of the i'th nonzero entry in row r of M. Let

$$x = M^{-1}b, \quad |x\rangle = \frac{x}{\|x\|}. \tag{B5}$$

Then, [15] construct an algorithm relying on Hamiltonian simulation that outputs the state $|x\rangle$ up to precision ϵ, with constant probability of failure (i.e., independent from the problem parameters), and makes

$$O(d\kappa\,\log^{2.5}(\kappa/\epsilon)) \tag{B6}$$

uses of $\mathcal{P}_M$ and

$$O(\kappa\sqrt{\log(\kappa/\epsilon)}) \tag{B7}$$

of $\mathcal{P}_f$; and has overall time complexity

$$O(d\kappa\;\; poly(\log(n'\,d\kappa/\epsilon))). \tag{B8}$$

Also in the same article, [15], the authors present the lemmas 15 and 16, which we use □

Lemma 1. [15] *Let $|\lambda\rangle$ be an eigenvector of $H = M/d$, M a $\bar{n} \times \bar{n}$ matrix with sparsity d. Let $\lambda \in (-1, +1)$ the*

corresponding eigenvector. Within the invariant subspace $\{T|\lambda\rangle, ST|\lambda\rangle\}$*, the walk operator W has block form*

$$\begin{pmatrix} \lambda & -\sqrt{1-\lambda^2} \\ \sqrt{1-\lambda^2} & \lambda \end{pmatrix}. \tag{B9}$$

In particular, this will have the consequence that

$$W^j T|\lambda\rangle = \mathcal{T}_j(\lambda) T|\lambda\rangle + \sqrt{1-\lambda^2}\,\mathcal{U}_{j-1}(\lambda)|\perp_\lambda\rangle, \tag{B10}$$

with $\mathcal{T}_j$ *is the jth Chebyshev polynomial of the first kind, and* $\mathcal{U}_j$ *of the second kind; and* $|\perp_\lambda\rangle$ *a state in* $\{T|\lambda\rangle, ST|\lambda\rangle\}$ *perpendicular to* $T|\lambda\rangle$*.*

Notice that this lemma is the origin of (32).

Finally, let us state the theorem of amplitude estimation, which we use through the text.

Theorem 6 (Amplitude estimation) [38]: *for any positive integer k, the algorithm amplitude estimation outputs an estimate* $0 \leqslant \tilde{a} \leqslant 1$ *of the desired amplitude a such that*

$$|a - \tilde{a}| \leqslant 2\pi k \frac{\sqrt{a(1-a)}}{J} + k^2 \frac{\pi^2}{J^2}, \tag{B11}$$

with success probability at least $\frac{8}{\pi^2}$ *for* $k = 1$*, and with success probability greater than* $1 - \frac{1}{2(k-1)}$ *for* $k \geqslant 2$*. J is defined as the number of times we need the implementations of the oracle that tells whether an element is marked, for amplitude estimation. Also, if* $a = 0$ *then* $\tilde{a} = 0$*, and if* $a = 1$ *and J even, then* $\tilde{a} = 1$*.* □

Appendix C. Probabilistic classical strategies to sample with certainties

In the main text we have focused on the particular problem of sampling the most relevant points to determine the distance between different classes in the context of classifying. Here we focus on a more general setting, where one is given an oracle scoring function s and wants to sample the points of the space with higher score, with exponentially high probability. We will see that some quadratic advantage is possible for non-deterministic functions s.

Imagine we have an m-dimensional space $\mathcal{S}$, which we discretise in n_0 points in one dimension such that in the end it contains $N = n_0^m$ points. This is for example what happens when we have the space of images of $n_0 = (n_p \times n_p)$ pixels, each displaying 256 possible values. If the image is in colour then to each pixels three such values ranging from 0 to 255 will be assigned indicating the coordinates (red, green, blue), so that the number of possible points or images in the space would be $N = (256)^{3n_0} = (256)^{6n_p}$.

Another example could be, in the setting of reinforcement learning, the range of policies parametrised by n parameters, each of which can again take a range of discrete values. The policy of an agent is a function that takes as input the state of the agent or the observation it makes, and outputs an action which seeks to maximize the reward the agent gets. In this case the space $\mathcal{S}$ would be the space of possible policies. As we can see in both cases the number of points in the space is exponential in the dimension d.

Suppose that we also have some kind of scoring function, which we shall call $s : \mathcal{S} \to \mathbb{R}$, which may or may not be deterministic, and is given to us as an oracle. The problem we aim to solve is finding a point which is in the top $1/C$ fraction of points attaining the highest score, with some exponentially high probability. That it, we want to find one point x_0 such that $P(s(x_0)\text{in top}1/C) = 1 - p$ with $p = \mathrm{e}^{-\beta}$. Thus, C and β are the parameters we want to vary.

If the function s is probabilistic, it will output a score when evaluated on point x that follows a probability distribution with average μ_x and variance σ, $\mathcal{D}(\mu_x, \sigma)$. In this case the objective is to achieve the same guarantees for the average value of the evaluation of x_0. In other words, $P(\mu_{x_0}\text{ in top }1/C) = 1 - p$ with $p = \mathrm{e}^{-\beta}$. Let us summarise then the problem.

Problem 2 (Sampling with statistical guarantees). Let $\mathcal{S}$ be an m-dimensional discrete space. Let $s : \mathcal{S} \to \mathbb{R}$ be a scoring function that is given to us as an oracle. The problem is to return a point x that, with probability $1 - \mathrm{e}^{-\beta}$ is in the percentile $p_C = 1 - 1/C$ of s. If s is non-deterministic, but rather $s(x) \in \mathcal{D}(\mu = \mu_x, \sigma^2)$, $\mathcal{D}$ a probability distribution, then return a point x such that μ_x is in the previously mentioned percentile.

In the main text we initially compared our strategy with the most Naive classical one: if we want certainty that the point we choose really is in the top $1/C$ that means having to sample $O(1/N)$ of them. However we could argue that an exponentially small probability of the point not being in the top $1/C$ of the

most interesting ones is in fact a better comparison against our algorithm. After all, our algorithm does also achieve exponentially small probability of choosing the wrong point.

This appendix aims to propose a quantum strategy that allows us to efficiently sample one such point even if variance σ is large when compared against the range of values that μ_x may take, and compare it against classical strategies. We will see that the quantum strategy obtains a quadratic speedup on some parameters, due to the use of amplitude estimation [38] and amplitude estimation.

Let us start with the most naive of the classical strategies. If we try to sample points from the space until we get certainty that the best point of those sampled is one of the seeked ones, the cost will be exponential even in the deterministic case, taking $O(N(1-1/C))$ evaluations of s.

But if we only want statistical guarantees then one can do much better. The first of the probabilistic classical strategies can be called 'greedy'.

C.1. The classical greedy strategy

Suppose that we are in the deterministic case. The greedy strategy consists on following process:

(a) Uniformly at random, sample a single point x from the entire space $\mathcal{S}$ of possible candidate points.
(b) Evaluate $s(x)$.
(c) If $s(x)$ is higher than any of the previously evaluated ones, substitute the previous maximum point by $x, s(x)$.

By hypothesis we want to get a point that is in the top $1/C$, what means that each time we sample a point, we have a $1/C$ probability of spotting one. As this process gets the best of c candidates, the probability that the best one is not in the top $1/C$ decreases as $(1-1/C)^c$. Complexitywise, this means that we want to find c such that

$$p > \left(1 - \frac{1}{C}\right)^c, \tag{C1}$$

with p the probability of outputting a wrong point. Thus

$$c > \frac{\log\ p}{\log\left(1-\frac{1}{C}\right)}. \tag{C2}$$

Obviously both the nominator and the denominator are negative. Let us analyse this complexity by parts. First notice that

$$\lim_{C\to\infty} \frac{\frac{-1}{\log\left(1-\frac{1}{C}\right)}}{C} = 1, \tag{C3}$$

what means that $\frac{-1}{\log\left(1-\frac{1}{C}\right)} = O(C)$, matching the complexity of our algorithm for the C variable. On the other hand, as we make $p \to 0$, $-\log p$ goes very quickly to infinity. However if we make p exponentially small, $p = \mathrm{e}^{-\beta}$, then $-\log p = -\log \mathrm{e}^{-\beta} = \beta$. And this would imply that $c = O(\beta C)$.

Let us now consider the case where s is not deterministic but rather follows a distribution $\mathcal{D}(\mu_x, \sigma)$. In such case the greedy procedure becomes more complicated.

Let us introduce some notation. Assume that we name the points $x_0, \ldots, x_c$ such that for the first evaluation $s(x_0) \geqslant \ldots \geqslant s(x_c)$. Also denote by m_i the number of times x_i has been evaluated, since we will need to evaluate several times the same x_i.

In order to analyse the case for non-deterministic function s, we will make use of the central limit theorem:

Theorem 7. *Central limit theorem. Let X be a random variable that follows some probability distribution with average μ and variance $\sigma^2 < \infty$. If we sample n times independently from such probability distribution, the sample average $\hat{\mu}$ will approximately follow, for n large, a distribution with average μ and variance σ^2/n.*

Now, suppose that we have called the oracle function s m_0 times for x_0, and m_1 times for x_1. That means that the averages of the distributions, $\hat{\mu}_0$ and $\hat{\mu}_1$, will follow distributions with averages μ_0 and μ_1 and variances σ^2/m_0 and σ^2/m_1. Notice that the probability distribution of the difference of two normal distributions is again a normal distribution with mean $\mu_0 - \mu_1$ and variance $\sigma^2/m_0 + \sigma^2/m_1$ [49]. Thus, the probability that in fact $\mu_0 \geqslant \mu_1$ is given by

$$\int_0^\infty \mathcal{N}(\mu_0 - \mu_1, \sigma^2/m_0 + \sigma^2/m_1), \tag{C4}$$

$\mathcal{N}$ the normal distribution. The previous expression is equal to

$$\left[\frac{1}{2}+\frac{1}{2}\mathrm{erf}\left(\frac{z-(\mu_0-\mu_1)}{\sqrt{\sigma^2/m_0+\sigma^2/m_1}}\right)\right]_0^{\infty}, \tag{C5}$$

with z the variable and erf the error function. Simplifying, as the error function is antisymmetric,

$$P_1=\frac{1}{2}+\frac{1}{2}\mathrm{erf}\left(\frac{(\mu_0-\mu_1)}{\sqrt{\sigma^2/m_0+\sigma^2/m_1}}\right). \tag{C6}$$

One would like this to be larger than $1-p'$ such that $1-p\leqslant(1-p')^c$ and $1-p=1-\mathrm{e}^{-\beta}$, so

$$\frac{1}{2}+\frac{1}{2}\mathrm{erf}\left(\frac{(\mu_0-\mu_1)}{\sqrt{\sigma^2/m_0+\sigma^2/m_1}}\right)\geqslant 1-p' \tag{C7}$$

implies

$$\mathrm{erf}\left(\frac{(\mu_0-\mu_1)}{\sqrt{\sigma^2/m_0+\sigma^2/m_1}}\right)\geqslant 1-2p', \tag{C8}$$

and

$$\sigma^2/m_0+\sigma^2/m_1\leqslant\left(\frac{(\mu_0-\mu_1)}{\mathrm{erf}^{-1}(1-2p')}\right)^2. \tag{C9}$$

Notice that when $p\to 0$, the denominator goes to infinity. Since $p'\geqslant 1-(1-p)^{1/c}$, that means that $\mathrm{erf}^{-1}(1-2p')=\mathrm{erf}^{-1}(-1+2(1-p)^{1/c})$. Then, if we take $c=O(C\beta)$

$$\begin{aligned}&\lim_{\beta\to\infty}\frac{\mathrm{erf}^{-1}(-1+2(1-\mathrm{e}^{-\beta})^{1/c})}{\sqrt{\beta}}=\\&\lim_{\beta\to\infty}\frac{\mathrm{erf}^{-1}(-1+2(1-\mathrm{e}^{-\beta}/c))}{\sqrt{\beta}}=1.\end{aligned} \tag{C10}$$

Notice however that the above expression does not depend on c. If $m_0=m_1$ then $\mathrm{erf}^{-1}(1-2p')=O(\sqrt{\beta})$ and

$$m_1=m_0\geqslant\frac{2\sigma^2}{(\mu_0-\mu_1)^2}O(\beta). \tag{C11}$$

Notice that we have to check this for every pair (m_0,m_i) for all $i\in(1,\ldots,c)$, and as we had said that $c=O(\beta C)$, the overall complexity is upper bounded by $\Omega(\beta^2C)$. This result is consistent even if $m_0\neq m_1$, as can be easily checked. In contrast, we shall see that using quantum procedures we can achieve a complexity $O(\beta C)$.

Another interesting point to extract from the previous equation is that the complexity is greatly reduced whenever $\sigma^2\ll(\mu_0-\mu_1)^2$. In particular, when $\sigma^2=0$ we recover the limit for the oracle function s being deterministic.

But one may argue that actually one may be able to generalise the strategy. With this we mean that the objective is checking that μ_0 is in the top $1/C$, not that it is better than any other μ_i. Thus, the way to think about it is the probability of at least one of the c points being in the top $1/C$, times the probability of such point being x_0; plus the probability of at least two points being in the top $1/C$ times the probability of x_0 being the second best of the c points sampled…Thus calling P_i the value of (C6) when taken for points x_0 and x_i,

$$1-p\leqslant\sum_{n=1}^{c}\binom{c}{n}\left(\frac{1}{C}\right)^n\left(1-\frac{1}{C}\right)^{c-n}\cdot\sum_{j_1<\ldots<j_{n-1}}\prod_{k=1}^{n}(1-P_{j_k})\prod_{i\neq j_1,\ldots,j_{n-1}}P_i. \tag{C12}$$

The idea here would be to calculate $c,m_0,\ldots,m_c$ fulfilling the previous equation while minimising $\sum_{i=0}^{c}m_i$. However, optimizing $m_1,\ldots m_c$ over the previous set can be complicated. This suggests using a different approach, that we review in the following section.

C.2. Threshold strategy

In the previous section we followed a greedy strategy that was quite efficient when the function s was deterministic, but became more involved when the output of s followed a distribution with variance σ^2 relatively big compared to the distance $(\mu_0-\mu_1)^2$.

Thus, in this last case, another good strategy could be

(*a*) Repeatedly sample uniformly at random points $x \in \mathcal{S}$, and apply $s(x)$, in order to calculate the percentile $1 - 1/C$ of the evaluation of $s(x)$, p_C with error ϵ and exponentially good probability $1 - p = 1 - \mathrm{e}^{-\beta}$.

(*b*) Find a single point x and evaluate $s(x)m_x$ times to make sure that, with exponentially high probability x is in the top $1 - 1/C$ quantile.

Thus the first step is estimating a percentile. If we sample n points $x_1, \ldots, x_n$ and evaluate $s(x_i)$ for each of them, such that $s(x_1) \leqslant \ldots \leqslant s(x_n)$. Then the best estimation of the value in the $1 - 1/C$ percentile is the weighted average between $x_{\lceil (n+1)(1-1/C) \rceil}$ and $x_{\lfloor (n+1)(1-1/C) \rfloor}$, with weights $|\lceil (n+1)(1-1/C) \rceil - (n+1)(1-1/C)|$ and $|\lfloor (n+1)(1-1/C) \rfloor - (n+1)(1-1/C)|$ respectively. To clarify, suppose for example that we sample 3 points $s(x_1) < s(x_2) < s(x_3)$ from a uniform distribution and we want to calculate the $1/3$ quantile. As the previous calculation and our intuition indicate, it should be the (weighted) average of x_1 and x_2, because that leaves $1/3$ of the points below it.

More generally, subindex i follows, for large n, a probability distribution $\mathcal{D}$ of p_C being x_i. Such probability distribution is a normal distribution with average $\mu = np_C$ and variance $\sigma^2 = np_C(1 - p_C)$. In other words, the variable

$$z = \frac{i - np_C}{\sqrt{np_C(1 - p_C)}} \tag{C13}$$

follows the normal distribution $\mathcal{N}(\mu = 0, \sigma^2 = 1)$. Since we want to estimate p_C with error $\epsilon' = \epsilon/C$ and exponentially high probability $1 - p$, this means

$$P\left(\frac{(1-\epsilon')p_C n - p_C n}{\sqrt{np_C(1-p_C)}} \leqslant z \leqslant \frac{(1+\epsilon')p_C n - p_C n}{\sqrt{np_C(1-p_C)}} \right) \geqslant 1 - p. \tag{C14}$$

The reason for choosing $\epsilon' = \epsilon/C$ is to be able to compare with the quantum algorithm. Simplifying,

$$P\left(-\epsilon' \sqrt{n} \sqrt{\frac{p_C}{1-p_C}} \leqslant z \leqslant \epsilon' \sqrt{n} \sqrt{\frac{p_C}{1-p_C}} \right) \geqslant 1 - p. \tag{C15}$$

Calculating the left-hand side of the equation, as $\mu_z = 0$ and $\sigma_z = 1$

$$\begin{aligned} \left[\frac{1}{2} + \frac{1}{2} \mathrm{erf}\left(\frac{z - \mu_z}{\sqrt{2}\sigma_z} \right) \right]_{-\epsilon' \sqrt{n} \sqrt{\frac{p_C}{1-p_C}}}^{+\epsilon' \sqrt{n} \sqrt{\frac{p_C}{1-p_C}}} &= \frac{1}{2} \left[\mathrm{erf}\left(\frac{\epsilon' \sqrt{np_C}}{\sqrt{2(1-p_C)}} \right) - \mathrm{erf}\left(\frac{-\epsilon' \sqrt{np_C}}{\sqrt{2(1-p_C)}} \right) \right] \\ &= \mathrm{erf}\left(\frac{\epsilon' \sqrt{np_C}}{\sqrt{2(1-p_C)}} \right) \geqslant 1 - \mathrm{e}^{-\beta}, \end{aligned} \tag{C16}$$

the last equal due to the error function being odd. This means we need the number of samples to grow like

$$n \geqslant 2 \frac{1 - p_C}{p_C} \frac{1}{\epsilon'^2} \left(\mathrm{erf}^{-1}(1 - \mathrm{e}^{-\beta}) \right)^2, \tag{C17}$$

where erf^{-1} is the inverse error function, and $(1 - p_C)/p_C = (C - 1)^{-1}$.

As

$$\lim_{\beta \to \infty} \frac{\mathrm{erf}^{-1}(1 - \mathrm{e}^{-\beta})}{\sqrt{\beta}} = 1, \tag{C18}$$

and $\epsilon' = \epsilon/C$, that means that $n = O(\epsilon^{-2} \beta C)$. As a second step we need to calculate the number of times we have to call the sample function s over a candidate in order to certify that with exponentially high probability it is above the percentile $1 - 1/C$. Using the central limit theorem again, the m-sample average $\hat{\mu}_x$ follows a normal distribution $\mathcal{N}(\mu_x, \sigma^2/m)$. So, given a threshold T which indicates the upper part of the confidence interval of the percentile $1 - 1/C$; and supposing $\mu_x > T$, we want

$$\int_T^\infty \mathcal{N}(\mu_x, \sigma^2/m) \geqslant 1 - p = 1 - \mathrm{e}^{-\beta}. \tag{C19}$$

Table C1. Complexities of different algorithms. Notice that the step of determining the threshold is dominating in the second and third strategies. However, there is also a difference in the step of determining a point over the threshold. Classically, the complexity would be $O(\beta C)$, whereas quantumly it would be $O(\epsilon^{-1}\beta\sqrt{C})$.

Complexity	Deterministic	Non-deterministic
Classical greedy	$O(\beta C)$	$\Omega(\beta^2 C)$
Classical threshold	$O(\epsilon^{-2}\beta C)$	$O(\epsilon^{-2}\beta C)$
Quantum threshold	$O(\epsilon^{-1}\beta C)$	$O(\epsilon^{-1}\beta C)$

The left-hand side can be written as

$$\left[\frac{1}{2}+\frac{1}{2}\mathrm{erf}\left(\frac{z-\mu_x}{\sqrt{2\,m}\sigma}\right)\right]_T^{\infty}=\frac{1}{2}-\frac{1}{2}\mathrm{erf}\left(\frac{T-\mu_x}{\sqrt{2\,m}\sigma}\right). \tag{C20}$$

From this it can be calculated that

$$m\geqslant\frac{2\sigma^2}{(\mu_x-T)^2}(\mathrm{erf}^{-1}(1-2\mathrm{e}^{-\beta}))^{-2}, \tag{C21}$$

and as

$$\lim_{\beta\to\infty}\frac{\mathrm{erf}^{-1}(1-2\mathrm{e}^{-\beta})}{\sqrt{\beta}}=1, \tag{C22}$$

we have $m=O(\beta)$. Over how many points x do we have to repeat this procedure until we make sure that x is over the threshold? As described just after (C3), one would need to repeat this over $O(C)$ points, although in practice it may be much less given the information we have of the previous steps. This step then has complexity $O(\beta C)$. Thus the overall complexity of this procedure is $O(\epsilon^{-2}\beta C)$.

C.3. Quantum threshold strategy

Can we do better if we use quantum methods? The answer is that we can get a quadratic advantage over some parameters: we will have an overall complexity of $O(\epsilon^{-1}C\beta)$.

Before explaining how to do it, we need to define the equivalent version of the probabilistic oracle s. In this case $s:|x\rangle\to\sum_{s(x)}\sqrt{p_{x,s(x)}}|x\rangle|s(x)\rangle$, where $p_{x,s(x)}$ are the probabilities and follow the distribution $\mathcal{D}(\mu_x,\sigma^2)$.

The required steps are the following

(*a*) Use bisection search, amplitude estimation [38] and the median lemma from [50] to estimate, with success probability $1-p=1-2^{-\beta}$ and error ϵ a threshold T for the percentile $1-1/C$ of the Image of s.

(*b*) Use amplitude estimation [38] to mark any $|x\rangle$ for which the amplitude over threshold $T+\epsilon$ is greater than $1/\sqrt{2}+\epsilon$. Use and the median lemma from [50] to achieve an exponential high success probability $1-2^{-\beta}$. The time complexity would be $O(\beta\epsilon)$.

(*c*) Use amplitude amplification [38] to find all such points in time $O(\sqrt{C})$.

Since the median lemma is not widely known, let us state it here without proof.

Lemma 2 (Median lemma) [50]. *Consider a sequence $\{\phi_k'\}$ for $k=1\ldots\beta$. Let the probability that ϕ_k' does not belong to (ϕ_L,ϕ_R) be smaller $\delta<1/2$. Then, the probability that the median of $\{\phi_k'\}$ falls out of (ϕ_L,ϕ_R) can be bounded above by*

$$p_{\mathrm{fail}}=\frac{1}{2}\left(2\sqrt{\delta(1-\delta)}\right)^{-\beta}\leqslant 2^{-\beta-1}. \tag{C23}$$

□

This lemma can be used to reduce, in linear time $O(\beta)$, the probability that amplitude estimation with error ϵ fails. On the other hand, the complexity of amplitude estimation is $O(\epsilon^{-1})$, for a given error ϵ.

This means that given an error tolerance ϵ/C, and a probability of failure $2^{-\beta}$ we can perform amplitude estimation with complexity $O(\epsilon^{-1}\beta C)$. Finally, to fully understand the step 1 above, we need to explain what we mean by bisection search. The idea is to propose a trial threshold T' and check whether the percentage of points for which $s(x)$ is above T' is greater or smaller than $1/C$. With that one may refine the initial guess of T' until we know that the percentage of points for which $s(x)>T$ is between $(1-\epsilon)/C$ and $(1+\epsilon)/C$ with exponentially high probability $1-\mathrm{e}^{-\beta}$. The cost of bisection search is well known to be logarithmic in the precision.

Overall we can see the complexity of each strategy in the table C1.

Appendix D. The norm of a vector after quantum linear algebra techniques

In the appendix A of [37] it is explained how to estimate the norm of the solution vector of the HHL algorithm, as we already indicated in the equation (21). However, at the end of the appendix they indicated that they were not sure how to estimate the norm of the solution using any other improvements to HHL such as [15, 35]. Here we explain how to estimate the norm of any vector $M|v\rangle$, for M any matrix that we can decompose as a linear combination of unitaries $M = \sum_i \alpha_i U_i$, and $|v\rangle$ a pure quantum state; or when we apply the Chebyshev approach indicated in the main text.

Firstly, let us briefly review the simplest case of LCU technique. Let $M = U_0 + U_1$. Then we can perform

$$|0\rangle|v\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|v\rangle. \tag{D1}$$

Now, we perform the so called multi-U operator $\sum_i |i\rangle\langle i| \otimes U_i$,

$$\frac{1}{\sqrt{2}}(|0\rangle U_0 + |1\rangle U_1)|v\rangle, \tag{D2}$$

and performing a second Hadamard over the first register we get

$$\frac{1}{2}\left[|0\rangle(U_0 + U_1) + |1\rangle(U_0 - U_1)\right]|v\rangle. \tag{D3}$$

Postselecting on measuring $|0\rangle$ on the first register we get a state proportional to $M|v\rangle$. To calculate the probability, let us write it in density matrix form

$$\begin{aligned}\rho = \frac{1}{4}\big[&|0\rangle\langle 0|(U_0 + U_1)|v\rangle\langle v|(U_0 + U_1)^\dagger \\ &+ |0\rangle\langle 1|(U_0 + U_1)|v\rangle\langle v|(U_0 - U_1)^\dagger \\ &+ |1\rangle\langle 0|(U_0 - U_1)|v\rangle\langle v|(U_0 + U_1)^\dagger \\ &+ |1\rangle\langle 1|(U_0 - U_1)|v\rangle\langle v|(U_0 - U_1)^\dagger\big].\end{aligned} \tag{D4}$$

To calculate the probability of measuring 0, we need to measure the norm of $P_0\rho P_0$, where $P_0 = |0\rangle\langle 0| \otimes \mathbf{1}$, taking the trace, which happens to be $\langle v|(U_0 + U_1)^2|v\rangle/4$. The probability of measuring a $|0\rangle$ is then $\langle v|(U_0 + U_1)^2|v\rangle/4$. This same calculation is performed to calculate the probability of measuring a $|0\rangle$ in the swap test.

Let us use this to generalise to the setting when $M = \sum_i \alpha_i U_i$. The first step, analogous to (D1) is applying an operator V that prepares the coefficients

$$|0\rangle|v\rangle \xrightarrow{V\otimes\mathbf{1}} \frac{1}{\sqrt{\sum_i \alpha_i^2}}\sum_i \alpha_i|i\rangle|v\rangle. \tag{D5}$$

This implies that we have to correct the norm we will measure with $\sqrt{\sum_i \alpha_i^2}$. Then, we apply the multi-U, $\sum_i |i\rangle\langle i| \otimes U_i$.

$$\frac{1}{\sqrt{\sum_i \alpha_i^2}}\sum_i \alpha_i|i\rangle U_i|v\rangle. \tag{D6}$$

Next we perform a Hadamard gate, $H|i\rangle = \sum_j (-1)^{ij}|j\rangle$ over the first register,

$$\frac{1}{\sqrt{\sum_i \alpha_i^2}}\sum_{i,j}(-1)^{ij}\alpha_i|j\rangle U_i|v\rangle. \tag{D7}$$

Using the same argument as we did in the simpler case, the amplitude of the first register being in state $|0\rangle$ is

$$A_0 = \frac{1}{\sqrt{\sum_i \alpha_i^2}}\left\|\left(\sum_i \alpha_i U_i\right)|v\rangle\right\|. \tag{D8}$$

Therefore, in the same way that the norm of the solution can be calculated using (21), we can calculate the norm of the solution in this case using

$$\|Mv\| = A_0\|v\|\sqrt{\sum_i \alpha_i^2}. \tag{D9}$$

Clearly, estimating a given amplitude requires the use of amplitude estimation procedure, with cost $O(\epsilon^{-1})$, and can be applied to the procedure of the quantum linear system algorithm of [15] for example.

Next we want to calculate the factor by which to correct the norm of a vector $|v\rangle$ after applying a Chebyshev polynomial according to (32). First notice that if $|v\rangle = |\lambda\rangle$, for $\lambda = 1$ the largest eigenvalue of M, then using (32) will result in staying the same state $|\lambda\rangle$, due to lemma 1. This implies that we will have to correct the norm of the solution by the largest eigenvalue $\lambda_{\max}$ that will divide M in order to ensure that the largest eigenvalue in $M/\lambda_{\max}$ is 1.

In such case, one can calculate the equivalent of (D9) as

$$\|Mv\| = A_0\|v\|\lambda_{\max}, \tag{D10}$$

where A_0 is the estimated amplitude of the correct state in (32).

In general though, $\lambda_{\max}$ is not known. In such case, [15] proposes making all entries in the matrix M smaller than $1/d$, d the sparsity. Then, they argue, the maximum eigenvalue will be in the interval $(-1, 1)$ and lemma 1 can still be used. In such case, we will use (D10) but substituting $\lambda_{\max}$ by the factor that makes all entries smaller than $1/d$. Notice also that when estimating the norm of the solution of the HHL algorithm, we supposed that the maximum eigenvalue was no larger than 1. Therefore, a similar treatment to what we do here should be also applied there.

ORCID iDs

P A M Casares https://orcid.org/0000-0001-5500-9115
M A Martin-Delgado https://orcid.org/0000-0003-2746-5062

References

[1] Lloyd S, Mohseni M and Rebentrost P 2013 Quantum algorithms for supervised and unsupervised machine learning arXiv:1307.0411
[2] Schuld M, Sinayskiy I and Petruccione F 2015 An introduction to quantum machine learning *Contemp. Phys.* **56** 172–85
[3] Biamonte J, Wittek P, Pancotti N, Rebentrost P, Wiebe N and Lloyd S 2017 Quantum machine learning *Nature* **549** 195–202
[4] Paparo G D, Dunjko V, Makmal A, Martin-Delgado M A and Briegel H J 2014 Quantum speedup for active learning agents *Phys. Rev.* X **4** 031002
[5] Melnikov A A, Nautrup H P, Krenn M, Dunjko V, Tiersch M, Zeilinger A and Briegel H J 2018 Active learning machine learns to create new quantum experiments *Proc. Natl Acad. Sci.* **115** 1221–6
[6] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I and Fergus R 2013 Intriguing properties of neural networks arXiv:1312.6199
[7] Goodfellow I J, Shlens J and Szegedy C 2014 Explaining and harnessing adversarial examples arXiv:1412.6572
[8] Schmidt L, Santurkar S, Tsipras D, Talwar K and Madry A 2018 Adversarially robust generalization requires more data *Advances in Neural Information Processing Systems* (Red Hook, NY: Curran Associates) pp 5019–31
[9] Sinha A, Namkoong H and Duchi J 2017 Certifying some distributional robustness with principled adversarial training arXiv:1710.10571
[10] Wong E and Kolter J Z 2017 Provable defenses against adversarial examples via the convex outer adversarial polytope arXiv:1711.00851
[11] Raghunathan A, Steinhardt J and Liang P 2018 Certified defenses against adversarial examples arXiv:1801.09344
[12] Khoury M and Hadfield-Menell D 2018 On the geometry of adversarial examples arXiv:1811.00525
[13] Giovannetti V, Lloyd S and Maccone L 2008 Quantum random access memory *Phys. Rev. Lett.* **100** 160501
[14] Rebentrost P, Mohseni M and Lloyd S 2014 Quantum support vector machine for big data classification *Phys. Rev. Lett.* **113** 130503
[15] Childs A M, Kothari R and Somma R D 2017 Quantum algorithm for systems of linear equations with exponentially improved dependence on precision *SIAM J. Comput.* **46** 1920–50
[16] Qin C, Martens J, Gowal S, Krishnan D, Dvijotham K, Fawzi A, De S, Stanforth R and Kohli P 2019 Adversarial robustness through local linearization *Advances in Neural Information Processing Systems* pp 13824–33
[17] Liu N and Wittek P 2019 Vulnerability of quantum classification to adversarial perturbations arXiv:1905.04286
[18] Lu S, Duan L-M and Deng D-L 2019 Quantum adversarial machine learning arXiv:2001.00030
[19] Wang L, Hu X, Yuan B and Lu J 2015 Active learning via query synthesis and nearest neighbour search *Neurocomputing* **147** 426–34
[20] Atlas L E, Cohn D A and Ladner R E 1990 Training connectionist networks with queries and selective sampling *Advances in Neural Information Processing Systems* pp 566–73
[21] Lewis D D and Gale W A 1994 A sequential algorithm for training text classifiers *SIGIR'94* (Berlin: Springer) pp 3–12
[22] Melville P and Mooney R J 2004 Diverse ensembles for active learning *Proc. of the 21st Int. Conf. on Machine Learning* (ACM) p 74
[23] Roy N and McCallum A 2001 Toward optimal active learning through monte carlo estimation of error reduction *ICML* (*Williamstown*) pp 441–8
[24] Dunjko V and Briegel H J 2018 Machine learning & artificial intelligence in the quantum domain: a review of recent progress *Rep. Prog. Phys.* **81** 074001
[25] Schuld M and Killoran N 2019 Quantum machine learning in feature hilbert spaces *Phys. Rev. Lett.* **122** 040504
[26] Pérez-Salinas A, Cervera-Lierta A, Gil-Fuster E and Latorre J I 2020 Data re-uploading for a universal quantum classifier *Quantum* **4** 226

[27] Zhao Z, Pozas-Kerstjens A, Rebentrost P and Wittek P 2019 Bayesian deep learning on a quantum computer *Quantum Machine Intelligence* **1** 41–51
[28] Farhi E and Neven H 2018 Classification with quantum neural networks on near term processors arXiv:1802.06002
[29] Broughton M *et al* 2020 Tensorflow quantum: a software framework for quantum machine learning arXiv:2003.02989
[30] Zoufal C, Lucchi A and Woerner S 2019 Quantum generative adversarial networks for learning and loading random distributions *npj Quantum Information* **5** 1–9
[31] Schuld M and Petruccione F 2018 *Supervised Learning with Quantum Computers* vol 17 (Berlin: Springer)
[32] Harrow A W, Hassidim A and Lloyd S 2009 Quantum algorithm for linear systems of equations *Phys. Rev. Lett.* **103** 150502
[33] Lloyd S, Mohseni M and Rebentrost P 2014 Quantum principal component analysis *Nat. Phys.* **10** 631–3
[34] Childs A M 2010 On the relationship between continuous-and discrete-time quantum walk *Commun. Math. Phys.* **294** 581–603
[35] Ambainis A 2012 Variable time amplitude amplification and quantum algorithms for linear algebra problems *STACS'12 (29th Symp. on Theoretical Aspects of Computer Science), LIPIcs* vol 14 pp 636–47
[36] Wang C and Wossnig L 2018 A quantum algorithm for simulating non-sparse hamiltonians arXiv:1803.08273
[37] Montanaro A and Pallister S 2016 Quantum algorithms and the finite element method *Phys. Rev.* A **93** 032324
[38] Brassard G, Hoyer P, Mosca M and Tapp A 2002 Quantum amplitude amplification and estimation *Contemp. Math.* **305** 53–74
[39] Kerenidis I and Prakash A 2017 Quantum recommendation systems *Proc. of the 8th Innovations in Theoretical Computer Science Conf.*
[40] Casares P and Martin-Delgado M 2019 A quantum ip predictor-corrector algorithm for linear programming arXiv:1902.06749
[41] Jackson D 1930 *The Theory of Approximation* vol 11 (Providence, RI: American Mathematical Society)
[42] Childs A M and Kothari R 2009 Limitations on the simulation of non-sparse hamiltonians arXiv:0908.4398
[43] Berry D W, Childs A M and Kothari R 2015 Hamiltonian simulation with nearly optimal dependence on all parameters *2015 IEEE 56th Annual Symp. on Foundations of Computer Science* (IEEE) pp 792–809
[44] Cortes C and Vapnik V 1995 Support-vector networks *Machine learning* **20** 273–97
[45] Arrazola J M, Delgado A, Bardhan B R and Lloyd S 2019 Quantum-inspired algorithms in practice arXiv:1905.10415
[46] Tang E 2018 A quantum-inspired classical algorithm for recommendation systems arXiv:1807.04271
[47] Kerenidis I and Prakash A 2016 Quantum recommendation systems arXiv:1603.08675
[48] Chakraborty S, Gilyén A and Jeffery S 2018 The power of block-encoded matrix powers: improved regression techniques via faster hamiltonian simulation arXiv:1804.01973
[49] Weisstein E W Normal difference distribution (http://mathworld.wolfram.com/NormalDifferenceDistribution.html)
[50] Nagaj D, Wocjan P and Zhang Y 2009 Fast amplification of qma arXiv:0904.1549